2013

Wireless Sensor Networks And Data Fusion For Structural Health Monitoring Of Aircraft

Tori Romell Patterson
North Carolina Agricultural and Technical State University

Wireless Sensor Networks and Data Fusion for Structural Health Monitoring of Aircraft

Tori Romell Patterson

North Carolina A&T State University

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Department: Computer Science

Major Professor: Dr. Albert Esterline

Greensboro, North Carolina

2013

School of Graduate Studies
North Carolina Agriculture and Technical State University
This is to certify that the Master's Thesis of

Tori Romell Patterson

has met the thesis requirements of
North Carolina Agriculture and Technical State University

Greensboro, North Carolina
2013

Approved by:

_____	_____
Dr. Albert Esterline Major Professor	Dr. Mannur Sunderesan Committee Member
_____	_____
Dr. Justin Zhan Committee Member	Dr. Gerry Dozier Department Chairperson

_____
Dr. Sanjiv Sarin
Dean, The Graduate School

Biographical Sketch

Tori Romell Patterson was born on July 19, 1980 in Baltimore, Maryland. He received his Bachelor of Science degree in Computer Aided Drafting and Design from North Carolina Agricultural and Technical State University in December of 2002. From the spring semester of 2010 until his graduation date, Tori has been funded under the NASA Center for Aviation Safety and has conducted research related to this field of study. He is a candidate for the MS in Computer Science.

Dedication

First and foremost, I would like to thank God for the many blessings and the strength given to me through the process of creating this master's thesis. I would like to dedicate this thesis to my parents; Reginald Patterson, for giving me much needed structure and direction while reaffirming my ability to achieve my goal, and Theresa Patterson, for providing love, an open understanding ear and allowing me avenue to vent. I also would like to dedicate this to my two brothers; Rashan Patterson, older brother, for providing an ever-present example of courage and drive, and Brandon Patterson, younger brother, for showing what can be accomplished through hard work, motivation and determination. To Tina, thank you for waking me up, keeping me motivated and pushing me to see this through to the end. To my friends and family, thank you all for your love, support, encouragement, understanding and advice. You helped me to weather the storm, stay the course and see this chapter to its end. I am forever in all of your depts. Without the prayers and support of my family and friends, my educational aspirations may not have been so easily achieved.

Acknowledgements

I would like to thank Dr. Albert Esterline for allowing me the opportunity to work on this project and introducing me to the broad areas of research. I would also like to thank William Wright for direction and guidance throughout my time on this project. I would also like to thank Yolanda B. Jones, Samantha Beal and Gina Bullock for their guidance, assistance and support on this project as well. I would not have accomplished as much progress on this project without all of you setting up the foundation for me to build off of and providing solid direction for where to take this project.

I would also like to acknowledge the support of NASA Center for Aviation Safety.

Table of Contents

List of Figures x

List of Tables xi

Abstract 2

CHAPTER 1 Introduction 3

CHAPTER 2 Background 5

2.1 Structural Health Monitoring 7

2.1.1 Technology.. 7

2.1.2 Data driven prognostics.. 8

2.2 Wireless Sensor Networks (WSN) 10

2.2.1 Mesh networks. 11

2.2.2 Wireless protocols. 13

2.2.3 Wireless motes and sensors. 24

2.2.4 TinyOS/nesC. 30

2.3 Agents 32

2.3.1 ACL 33

2.3.2 Multiagent systems. 38

2.3.2 Contract Net Protocol. 41

2.3.3 Gaia.. 43

2.3.4 Software.. 45

2.3.4.1 JADE. 45

2.3.4.2 Jess. 46

2.3.4.3 Agilla. 49

2.4 Data Fusion 50

2.4.1 Python 52

2.4.2 Machine Learning.. 54

2.5 Acoustic Emissions 56

CHAPTER 3 Prototype Implementation 59

3.1 Phase1 Implementation 59

3.1.1 Contract Net Protocol. 60

3.1.1 Feature 63

3.1.2 Source to base station data flow. 64

CHAPTER 4 Discussion 67

4.1 Agent Programming 67

4.2 JADE 68

4.3 Wireless Sensor Network 71

CHAPTER 5 Conclusion 73

5.1 Future Work 74

References 78

Appendix A 83

Appendix B 88

Appendix C 90

Appendix D 92

Appendix E 94

Appendix F 95

Appendix G 97

Appendix H 98

Appendix I 100

Appendix J 103

Appendix K 106

Appendix L 109

Appendix M 110

Appendix N 111

Appendix O 113

List of Figures

Figure 1. Data Driven Prognostic Strategies 9

Figure 2. Plane Wireless Sensor Network. 10

Figure 3. Wireless Mesh Network 12

Figure 4. Scatternet 14

Figure 5. Overlapping Multiagent Systems 40

Figure 6. FIPA Contract Net Interaction Protocol 42

Figure 7. Complete Analysis and Design Model 44

Figure 8. Data Fusion Model. 51

Figure 9. Acoustic Emission 57

Figure 10. Proposed Phase1 SHM Architecture. 59

Figure 11. Date Flow. 60

Figure 12. Sniffer Agent GUI of Contract Net Protocol. 62

Figure 13. LabView Simulated Acoustic Emission VI (Configuration File). 64

Figure 14. Simulated Acoustic Emission Sine Wave. 65

Figure 15. Data Agent Gaia Role Schema 69

Figure 16. Future System Architecture. 74

List of Tables

Table 1 Comparison of the Bluetooth, Zigbee, and WI-FI Protocols 22

Table 2 Comparison of the Imote2, Cricket and Sunspot Devices 27

Table 3 KQML Performatives 34

Table 4 FIPA-ACL Performatives 36

Abstract

This thesis discusses an architecture and design of a sensor web to be used for structural health monitoring of an aircraft. Also presented are several prototypes of critical parts of the sensor web. The proposed sensor web will utilize sensor nodes situated throughout the structure. These nodes and one or more workstations will support agents that communicate and collaborate to monitor the health of the structure. Agents can be any internal or external autonomous entity that has direct access to affect a given system. For the purposes of this document, an agent will be defined as an autonomous software resource that has the ability to make decisions for itself based on given tasks and abilities while also collaborating with others to find a feasible answer to a given problem regarding the structural health monitoring system. Once the agents have received relevant data from nodes, they will utilize applications that perform data fusion techniques to classify events and further improve the functionality of the system for more accurate future classifications. Agents will also pass alerts up a self-configuring hierarchy of monitor agents and make them available for review by personnel. This thesis makes use of previous results from applying the Gaia methodology for analysis and design of the multiagent system.

CHAPTER 1

Introduction

For years, man has strived to make more comfortable and convenient homes, vehicles, and structures to ease the burden of everyday life. From our humble beginnings living in caves and hobbles made of sticks and leaves, to our more current brick, mortar and vinyl-sided houses, all our advances have been to creating structures with more stability and longer life.

There is only so much that can be done through the cultivation of newer materials for use in structures alone. We must also develop better techniques of studying the structural integrity of the structures themselves as well so that we can better identify possible breaking points or weaknesses in the structure sooner, allowing us more time to handle the breakdown of a structure in whole or in part. This is where the notion of structural health monitoring comes into play.

Structural health monitoring becomes increasingly vital in dealing with structures designed to transport groups of people over varying distances. With vehicles designed for urban transport, systems have been designed to test different facets of the engine, the breaking and suspension system, as well as more modern systems designed to take aspects of the vehicle's current environment into account to perform tasks such as automated parallel parking, collision detection or suspension update and realignment to suit the needs of varying environments.

The architecture and design discussed in this document is a combination of several structural health monitoring techniques and technologies together into one system for use within aeronautical vehicles. In aeronautics, physical weight becomes a major concern for the overall integrity and viability of the vehicle as added weight is multiplied by influences such as gravity, speed and wind resistance. To this end, the structural health monitoring system design will involve the use of wireless sensors controlled by utilizing autonomous programs in order to sense

the formation of cracks that pose a possible immediate or future threat to the health of the overall system and its passengers by characterizing acoustic emission events as possible threats to the overall structure. The system will also be used to store data from past events so that users are able to review them at a later time.

CHAPTER 2

Background

This chapter, a brief overview of some of the technologies and components addressed in the design is provided in order to give a better understanding of the system being implemented. This discussion will begin with a review of some key points regarding structural health monitoring. The first topic discussed will review structural health monitoring applications and techniques. Some of the currently implemented methods of structural health monitoring are discussed as well as some of the past and current challenges.

In the second section, the discussion turns to the review of different aspects of wireless sensor networks. A wireless sensor web is utilized in order to collect and pass data throughout a monitoring system. In this design, a sensor web is utilized as a network of nodes for capturing data and important alerts for both diagnosis and prognosis of the health of a structure. This section also describes the two main aspects of sensor networks that the design chiefly capitalizes on. The first of these aspects is the ability for the network to be collaborative. This is due to the ability of the sensor web to pass data and communications back and forth between nodes as well as the ability to intelligently collect and share data. This allows for goals to be broken down into sub-goals that can then be processed and achieved collaboratively. The second key factor that the sensor network is able to capitalize on is its ability to integrate an accessible web into the architecture. This is accomplished through web services, which allow for system monitoring as well as updates to be made remotely.

In this, the sensor web is able to be accessed and controlled remotely through the use of web services. In this section we will also review some of the researched sensor technology as well as the use of the Imote2 sensor for use within the network. This will also include brief

discussion of the programming language nesC, which runs on the TinyOS operating system. A brief overview of wireless sensor network architectures will also be discussed in an attempt to provide a basic understanding of the network architecture.

The next section addresses the need for this collaborative and accessible sensor network to be autonomous. The third section discusses the inclusion of a multiagent system for both autonomous control and self-monitoring of the network to answer this need. The addition of the multiagent system provides the system with the previously discussed collaborative abilities. In this section is also discussed the use of the Contract Net Protocol for control of collaboration among agents and the contracting and subcontracting of tasks throughout the network of agents. Also in this section we discuss aspects of Gaia, an agent-oriented analysis and design methodology, which is utilized to create dependable multiagent systems. This third section also discusses some of the software used to create this system of agents, Java Agent Development Framework (JADE) and Jess, a rule based engine utilized, in this application, for providing the agents with a level of intelligence.

These subjects are discussed in order to give a foundational view of the proposed work. In order to create a feasibly dependable system that is both robust and autonomous, having the ability to self-heal, configure, reconfigure and adapt, certain principles must be taken into account. This will better ensure that once the system is deployed in a live environment, it will be better able to autonomously adapt and conform to a given situation as well as react in as ideal a fashion as possible.

The fourth section of this chapter focuses on the topic of machine learning as well as data fusion. The correlation between the Joint Directors of Laboratories (JDL) data fusion model and the herein discussed implementation utilized for structural health monitoring is drawn in order to

break down the levels of data fusion. Python and the NumPy and SciPy libraries are discussed in this section to provide an understanding of the software utilized for data processing as well as the advantages and present foreseeable challenges. This section will also discuss the subject of pattern recognition as well as machine learning techniques.

2.1 Structural Health Monitoring

In a 2001 article on structural health monitoring from the United States Air Force Office of Research, structural health monitoring is described as the process of implementing a damage detection and characterization strategy for engineering structures to provide early warnings of unsafe conditions by using real-time data [47]. The aim of structural health monitoring is to provide a diagnosis of the current state of a structure as well as a prognosis of what is to come of the structure in the future given the diagnosis as well as current and future perceivable influences, both environmental and man-made. This process of determining the current integrity of a structure as well as possible future states has a very long history.

2.1.1 Technology. There is a host of technologies utilized to insure the structural integrity of large structures. These technologies are designed to detect minute fluctuations of a structure's integrity through capturing data regarding vibrations, crack delineations and light refraction. Some of the more widely utilized technologies include fiber optic sensors, active ultrasonic sensors, passive acoustic emission sensors and wireless sensor networks, which we discuss in a later section. Fiber optic sensors utilize beams of light produced from a laser to transmit light through the fiber. Oscillations to the laser stream that are functions of the temperature and stress of both the fiber optic and, by relation, the state of the monitored area are measured in order to either sense events or transmit data. Active ultrasonic sensors, similar to radar or sonar, evaluate attributes of a given target by interpreting echoes given off from radio

waves and or sound waves, respectively, by generating a high frequency sound wave and evaluating the echo which is sent back and then received by the sensor. Passive acoustic emission sensors are utilized to monitor and detect the ultrasonic waves that materials produce when cracks appear that could possibly lead to structural failure by utilizing triangulation techniques on an array of acoustic emission sensors to determine the location of the crack.

2.1.2 Data driven prognostics. Conventionally, algorithms used for damage propagation depend on physics-based failure mechanisms [17]. Instead, one can use data-driven tactics when sufficient quality test data is present that maps out the damage space. One purpose of this type of research is to measure the trade-offs that come from the amount of data needed, the computational speed displayed, the ability to support ambiguity management, and the prediction accuracy.

A central concern encountered in making a meaningful prediction that is as accurate as possible is to account for various kinds of uncertainties arising from different sources such as measurement noise, process noise, inaccurate process models, etc. Long-term prediction of the amount of time until failure involves large-grain uncertainty that must be characterized effectively and managed efficiently. For instance, as more data pertaining to past damage propagation and future use is made available, new means to narrow the uncertainty bounds must be devised. Metrics for prognostic performance should take the width of the uncertainty bounds into account. It is critical to select techniques that are able to take care of these issues while also providing damage trajectories. In many instances, a single technique is not sufficient to handle these unforeseeable issues, and so multiple techniques must be combined in order to better manage the uncertainty.

There are multiple strategies for tackling the process of learning. Some of these strategies are discussed later in the machine learning section. In the architecture discussed here, the raw data will be first reduced to features before being processed by classifiers created from machine learning algorithms. Figure 1 illustrates two strategies for addressing the learning

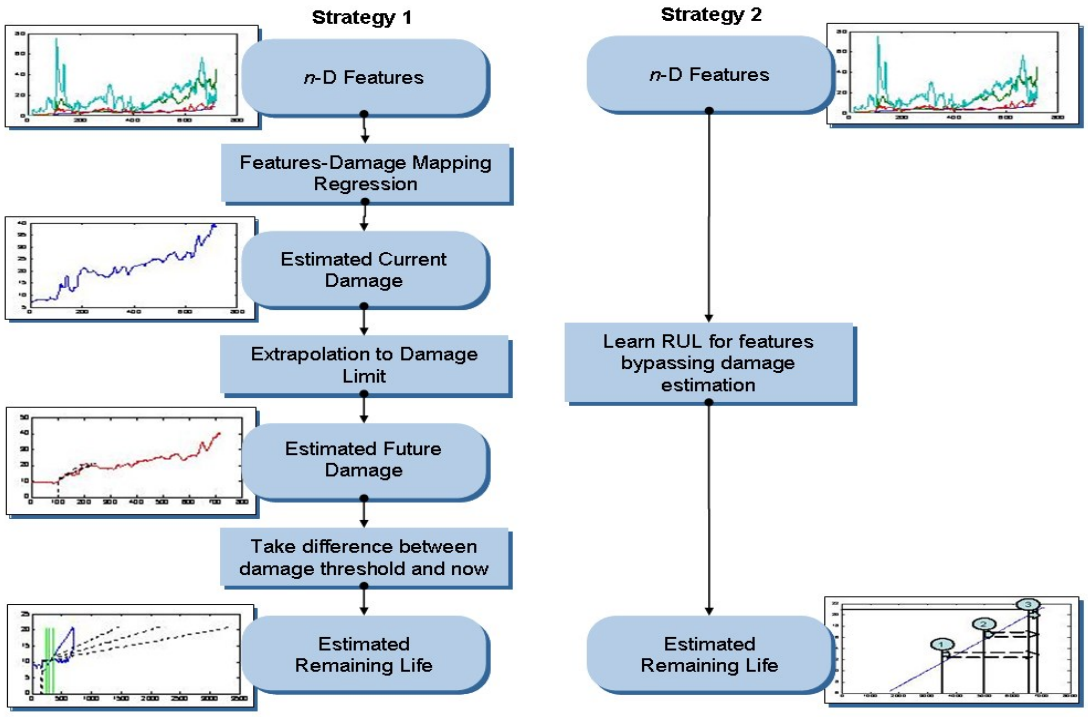


Figure 1. Data Driven Prognostic Strategies [17].

process. The first strategy begins with first mapping the dimensional features into a one-dimensional health or damage index using dimension reduction to estimate the current level of damage. Once the dimensions have been reduced, extrapolation to some predetermined damage (or health) limit allows one to compare the current level of damage to the damage limit in order to estimate the remaining useful life (RUL). In the next strategy, matching is performed directly on the n-dimensional features to find correlations between the current feature vectors and predetermined characterizations of damage determined through machine learning in order to ascertain the RUL of the structure.

2.2 Wireless Sensor Networks (WSN)

A wireless sensor network, such as the one shown in Figure 2, consists of two or more autonomous sensors utilized to *monitor* physical or environmental conditions, such as light, temperature, acoustic emissions, vibration, pressure, motion or pollutant [33]. The sensors are spatially distributed and positioned in a manner so that they are able to cooperatively pass their data through the network to a main location.

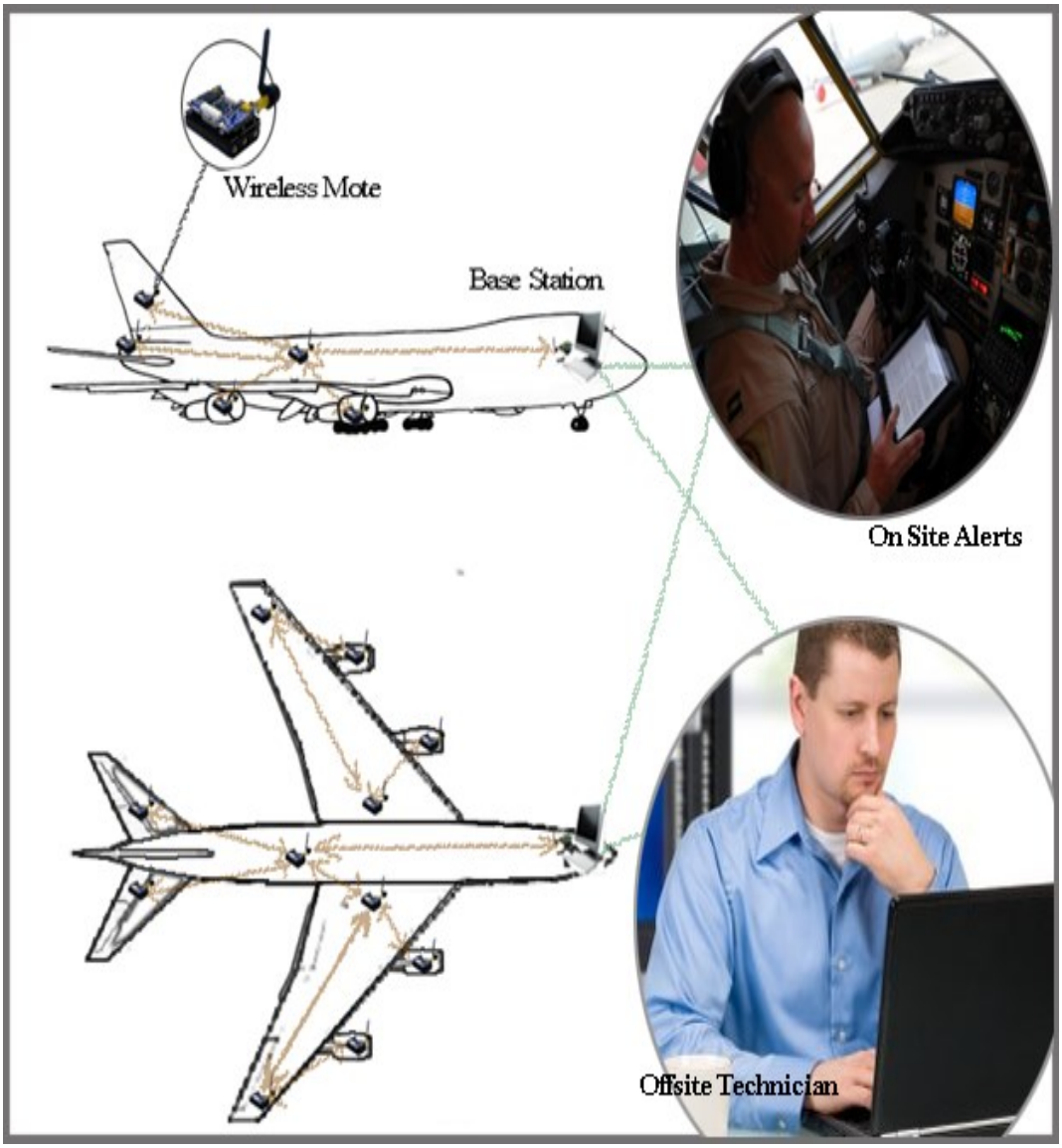


Figure 2. Plane Wireless Sensor Network.

This enables better control and utilization of data across a more robust network. Initially, development of wireless sensor networks was brought about for use in military applications such as battlefield surveillance and tactical reconnaissance. At present, the use of wireless sensor networks has been in many different applications, such as industrial process monitoring and control, and machine health monitoring.

Wireless sensor networks are made up of wireless sensor nodes. The number of nodes in a network can vary from two or more nodes based on the overall need. Each node is connected to at least one other to enable data transfer across the network. Nodes typically have several parts in common: a radio transceiver with an internal antenna or connection to an external antenna, an electronic circuit used for interfacing with sensors, an energy sources, which is some sort of battery or an embedded device for energy harvesting, and a microcontroller. Sensors vary in size and functionality. They can be as bulky as a shoebox, such as the lightweight fiber optic sensor created by NASA's Dryden Flight Research Center, or smaller than a grain of rice by using nano-scale technology like that utilized in the Nanosensor for bomb detection. The main utilized resources of the nodes are energy, memory, and communications bandwidth. These must be monitored to make sure that the systems limits are not exceeded.

The architecture, design, and implementation of any structure are the main determining factors in maintain the integrity of it. Not only is the creating a solid structural design of utmost importance but, to some, it is considered an art form. The term *system architecture* relates to the conceptual model that defines the behavior and overall structural design of a system. In the next section, the traits and characteristics of the mesh network architecture are reviewed.

2.2.1 Mesh networks. Wireless mesh networking is an emerging technology that brings us a step closer to a seamlessly connected world. Wireless mesh networks can utilize

existing technology inexpensively in order to effectively connect entire cities. Traditional sensor networks utilize a number of wired sensors affixed to a structure and linked into a base station. As depicted in Figure 3, in a wireless mesh network, the network connections can be spread out among countless wireless mesh nodes that communicate with each other via their programming in an attempt to share the network and cover a larger area more robustly. The sensors nodes that make up the mesh are equipped with small radio transmitters that function much the same as a wireless router. In this case, the nodes utilized the common WIFI standards of 802.11a, b and g

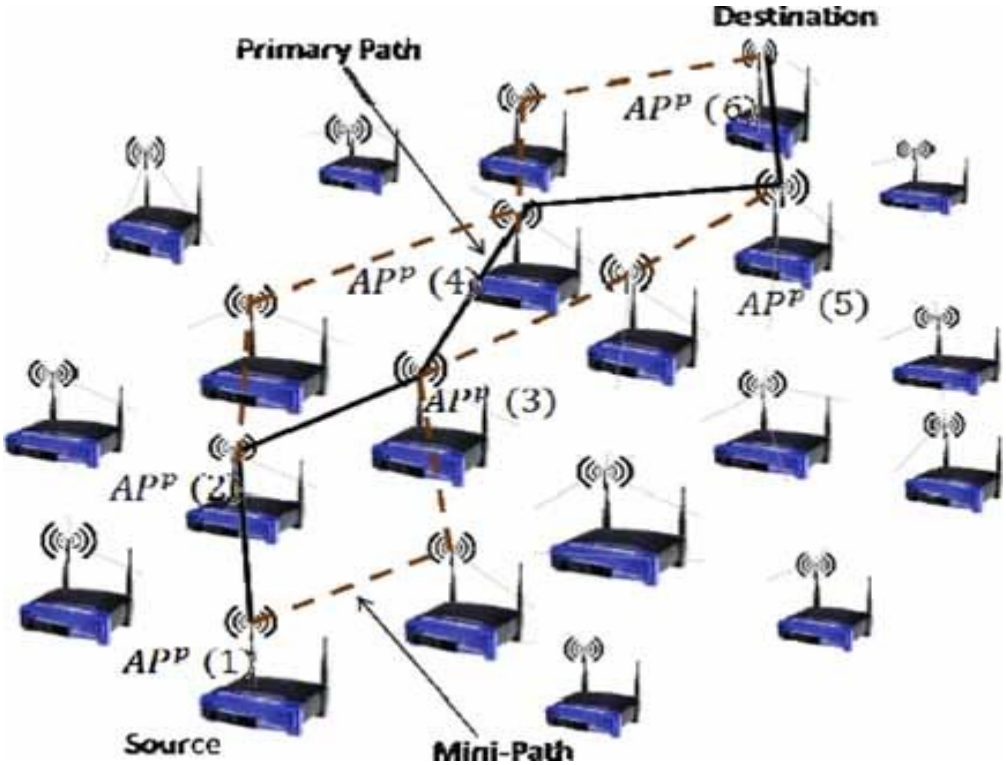


Figure 3. Wireless Mesh Network [45].

to send communications wirelessly between each other as well as back and forth to the user.

Nodes are usually programmed with software that gives them rules for interaction within the network. Utilizing a multihop configuration, data travels across the network from point to point by hopping wirelessly from on node in the mesh network to the next. The quickest and safest route is autonomously chosen for data to travel. One of the most advantageous assets of

using a wireless mesh network as opposed to a wired network is that there are no wires. Because of this, the time and manpower it normally takes to check for faulty hardware diminishes as a technician would only need to check the sensors themselves instead of the sensors and all the wires that connect them together. The wireless mesh network would also benefit from having less overall weight due again to the lack of wires.

In a wireless mesh network, only the base station node would need to be physically connected, using wires, to a computer terminal for processing information and connecting to the Internet. This node passes data back and forth wirelessly to the nodes closest to itself. Those nodes would then pass the information out in a ripple effect until all nodes that needed access to the information have been given the information for processing. The more nodes in the network, the further the network can spread and the faster the data can be passed within it.

2.2.2 Wireless protocols. The 802.x standards created by the Institute of Electrical and Electronics Engineers (IEEE) comprise a listing of networking standards that states physical layer specifications for data transfer technologies from Ethernet to wireless [36]. The physical layer contains basic networking hardware transmission technologies of a network [37]. The most common protocols in the 802 family are the 802.15.1 Bluetooth standard, the 802.11a/b/g Wi-Fi standard, and the 802.15.4 ZigBee standard. The IEEE protocols define standards for the physical layer and the medium access control (MAC) sublayer of the data link layer. The MAC sublayer is the data link sublayer responsible for providing addressing and channel access control mechanisms that make it possible for multiple network nodes or terminals to communicate within a multiple access network that incorporates a shared medium such as an Ethernet.

The Bluetooth standard of 802.15.1 relates to wireless radio systems utilized in short-range and inexpensive devices to replace previously hardwired computer devices with wireless ones [ptcl04]. Many devices such as the mouse, keyboards, joystick and printers have incorporated this functionality. This type of application is known as WPAN or wireless personal area network. There are two connectivity types defined in Bluetooth. These are piconet and scatternet, as depicted in *Figure 4*. In the piconet, one Bluetooth device serves as a master in the net and one or more other Bluetooth devices serves as slaves for that net. The clock of the master device is used to synchronize all of the other devices. Only the master is able to speak to all of the slave devices. A scatter, as depicted in *Figure 4*, is a combination of operational piconets which overlap in time and space. In such a network, the master node of one piconet may be a slave node in another.

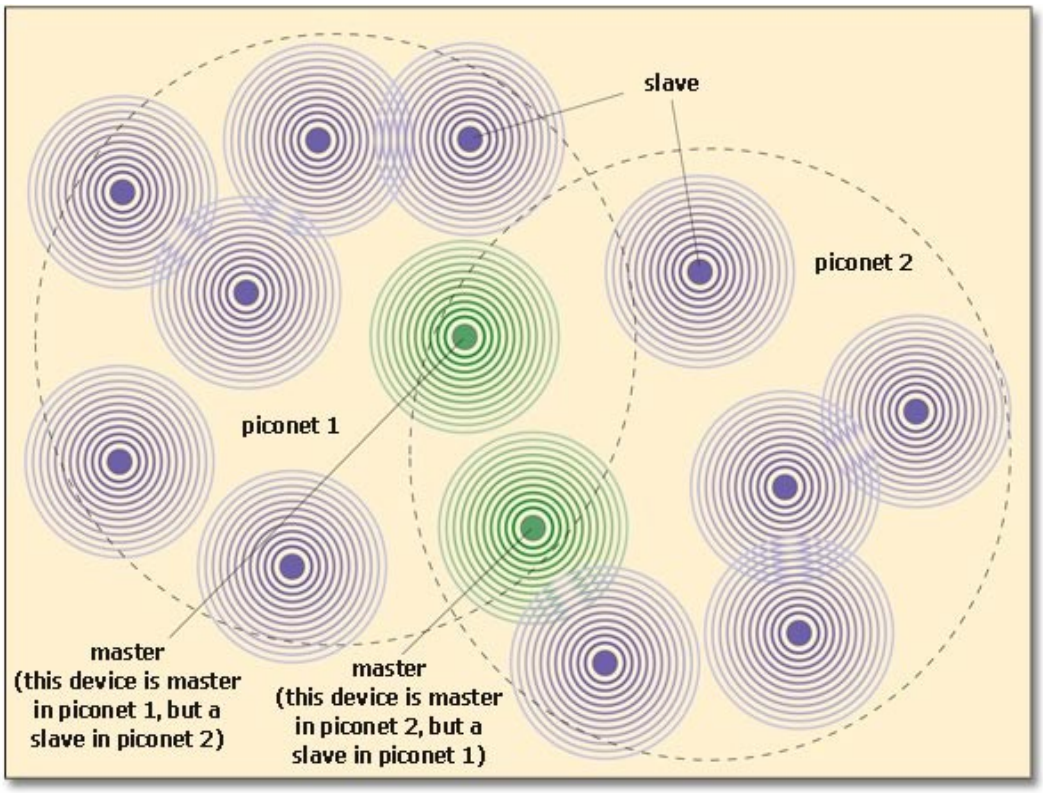


Figure 4. Scatternet [46].

The wireless fidelity or, Wi-Fi, standard 802.11a/b/g is a listing of protocols and standards relating to wireless local area networks [35]. 802.11 designations “a”, “b”, and “g” refer to amendments that have been added on to the original IEEE 802.11 specification. 802.11a added the ability to use a higher data rate of up to 54 Mb/s using the 5 GHz band. 802.11 b extended the throughput up to 11 Mb/s while still using the same band of 2.4 GHz. 802.11g is a combination of both 802.11a and 802.11b, allowing throughput up to rates of 54Mb/s while functioning in 2.4 GHz bands. These standards allow for users to access the Internet at broadband speeds while either connected to access points or in ad hoc mode, connecting to and communicating directly with wireless devices. The term “ad hoc” refers to a local area network (LAN) that is formed without pre-planning and only for as long as it is needed. The 802.11 design includes components that interact together to provide a wireless LAN that transparently supports station mobility to upper layers.

The 802.15.4 ZigBee wireless protocol is an IEEE standard used for specifying the media access control and physical layer for low rate wireless personal area networks. This is mainly used for supporting simple devices that consume small amounts of power and operate typically in the in a personal operating space of 10m. ZigBee provides multi-hop capabilities, allowing communication between two end nodes to be passed via a number of intermediary nodes with functions to relay information from one location to another, self-organizing capabilities and reliable mesh networking, with a long battery lifetime. Typically, there are two types of ZigBee protocol devices, RFD (Reduced Function Devices) and FFD (Full Function Devices). RFD’s or reduced-function devices are devices that do not need to send large amounts of data. Normally an RFD is utilized in applications that are simple in nature, such as a passive infrared sensor or a power switch. Because of this, RFD is able to be implemented using very minimal

resources and memory. RFDs are only able to communicate with one FFD at a time. FFD's or full-function devices are able to operate in three different modes. The first mode is the mode of a coordinator. The coordinator is an FFD with network device functionality which delivers coordination and other services to the network. The second mode is PAN (Personal Area Network) coordinator. The PAN coordinator is the device responsible for the formation of the ZigBee network. This PAN coordinator decides the PAN ID for the network and always designated with an address of 0. The last mode is as an RFD or FFD [24]. A FFD is able to communicate with other FFD's as well as RFD's. Once the FFD is activated for the first time, it is able to establish its own network by becoming the PAN (Personal Area Network) coordinator. A PAN ID, unique within the range of the radio sphere of influence, is designated for the network and other FFD's or RFD's can be allowed access. This allows the user to create an interconnected mesh network of devices for passing data.

When comparing wireless data transfer protocols, there are a few key attributes that should be discussed. These attributes control the amount of data that can be transferred, the distance that data can be successfully transferred, the amount of power to transfer that data and the security options available for securing that data. These options must be reviewed to decide the best technology for the needs of a system.

The first characteristic to be discussed is the *frequency band*. Frequency bands are groupings of radio frequencies that are used by mobile networks to communicate. There is a direct correlation between the frequency band and the wavelength. The researched protocols operate on frequencies that range from ultrahigh frequency, ranging from 300MHz to 3000MHz, to extremely high frequency, which ranges from 30GHz to 300GHz. The higher the frequency, the less atmospheric noise will cloud the data being transferred.

The *max signal rate*, similar to data signaling rate, is the maximum rate of bits per second at which binary information is able to be transferred in a given direction between users over a telecommunications system. The rate is based on data being transferred under conditions of continuous transmission and no overhead information. A higher maximum signal rate means that larger amounts of data could potentially be transferred at a faster rate than that of data transferred using a protocol of lower maximum signal rate.

Nominal range refers to the distance at which data can be successfully communicated. In the case of the researched protocols, it ranges from 10 meters to 100 meters. This can be an extremely important issue when deciding on wireless transfer protocols. If a proposed system will be enclosed in a relatively small area such as a vehicle, then a Bluetooth device could be sufficient as it allows for passing data up to 10 meters. If sensors will be dispersed over a larger area, sensing seismic patterns in Yellow Stone National Park for instance, it may be a better idea to use ZigBee or Wi-Fi to pass the data as they both allow for passing data up to 100 meters.

Nominal TX power is a measurement of the transmission output of a device. The measurement is represented in dBm or decibel milliwatts. This relates to the amount of power used by the device to transfer data. As some sensor systems are hard wired to a power source while others are wireless, this is an issue worth noting. In the case of wireless sensor network systems, many times the sensors are battery operated. Because usually sensors consume the largest amount of power when communicating data across the network, it is important to bear in mind the amount of power it takes to transfer that data, especially in cases where large amounts of data are expected to be transmitted. Sensors that are not currently sensing or communicating data can be placed in a sleep mode to conserve power.

Bandwidth is the measurement of the width of a frequency band. The bandwidth is measured in hertz. This is a measurement used for devices that provide communication abilities over distances. The *channel bandwidth* of a given wireless protocol is used to state the bandwidth at which data can be transferred on the communication channel that a device is currently communicating on.

A *coexistence mechanism* allows data sent in the designated protocol to be communicated to a device that operates with another protocol. This mechanism would allow, for example, data sent via a Bluetooth device to be received on a Wi-Fi device. Although, in the majority of cases, data will be communicated from one sensor to another using like technology, in some cases, data will need to be passed to a different type of device that may not have access to wireless protocols utilized in the main system. Coexistence mechanisms allow data to be successfully communicated across the network in such a case. There are several coexistence mechanisms used to transfer data from one protocol to another. *Adaptive Frequency Handling* (AFH), utilized by Bluetooth devices, attempts to use the good frequencies by avoiding the bad frequency channels. Picking the good frequencies is done by using a complemented mechanism for detecting channels with high traffic and directing data to channels with the least congestion. In *Dynamic Frequency Selection* (DFS), utilized by both ZigBee and Wi-Fi devices, radar signals are detected that must be protected against 802.11a interference. Once the harmful signals are detected, the 802.11a operating frequency is switched to one that is not experiencing interference by radar systems.

The *basic cell* refers to the basic network setup that can be created using the architecture of a given protocol. A network is made up of two or more devices linked together, either wirelessly or through tethered means, to communicate data. Different protocols allow devices to

be situated in different designs to achieve an optimum data transfer environment. The concept of a *maximum number of cell nodes* is used to specify the maximum number of devices that can be assigned to a basic cell. Bluetooth devices, as previously discussed, use a piconet architecture. The basic architecture used by ZigBee devices is the *star* architecture. The star architecture consists of a coordinator and a set of end devices that are only able to communicate information to each other though the coordinator. The basic architecture used by Wi-Fi devices is the basic service set (BSS). BSS, also utilized by the IEEE 802.11 WLAN architecture, is a set of stations that communicate with each other. If there are too many nodes for one network, then multiple basic networks can be created.

The *extension of a basic cell* refers to network architectures larger than that of the basic cell. In general, the extension of a basic cell is a network that builds upon the architecture of the basic cell. This type of cell can be achieved through the linking of two or more basic cells by introducing intermediary or proxy devices to link the different cells. An extension network can also be achieved through the overlapping of two or more networks. In the case of a network with overlapped basic cells, the master node in one cell could be considered the slave node in another. As previously discussed, the Bluetooth protocol utilizes the Scatternet architecture for extension of the basic cell. Devices utilizing the ZigBee protocol utilize the *cluster tree* architecture for extended cells. In a cluster tree architecture, as defined by the IEEE 802.15.4 standard, a global network coordinator is set as the root of a tree network whose nodes are able to act as local coordinators of their respective operating spaces. Each of these local coordinator nodes can be associated with a set of slave devices that are directly connected to them. Networks utilizing Wi-Fi devices in an extended basic cell will utilize the extended service set (ESS) architecture. ESS consists of a set of two or more BSSs, interconnected using a *distribution system* (DS) and local

area networks (LAN) that are linked together to appear as a single BSS. The distribution systems allow access points to communicate with each other and determine how traffic flows from one BSS to another. In order to boost the aggregate throughput, the BSSs included in the ESS can be set up to communicate on different channels. With data being communicated across varying networks, steps must be taken to ensure the security of the data.

Security is a very critical issue in any software application. Because many times data transferred across a network is of a sensitive nature, security has to be addressed to ensure that data is not able to be accessed or corrupted during transmission by unintended parties. *Encryption* is a method used in security for modifying data in such a way that only the sender and designated receiver are able to review it. The encryption process would normally involve the data being transformed into an incomprehensible state for communication and sent to the intended receiver with a key sent separately for decrypting it back into its normal state so that it can be reviewed or utilized. Data encrypted on Bluetooth devices utilize a technique called *stream cipher*. Stream cipher involves encrypting each bit of a data stream individually using a pseudorandom cipher character which is generated using a shifting seed value. The stream cipher is also known as a state cipher because encryption is dependent on the current state. Devices utilizing the ZigBee protocol utilize a form of encryption known as *advanced encryption standard (AES) block cipher*. AES, currently utilized by the U.S. government, is a symmetric-key algorithm. Symmetric-key encryption algorithms use the same key for encryption as well as decryption. In AES, blocks of data are encrypted and decrypted. AES has a fixed block size of 128 bits. Wi-Fi devices are more flexible than both Bluetooth and ZigBee devices as they are able to use both stream cipher and or AES for encryption of data. On top of securing the data being transferred, there has to be a way to verify that the sender and receiver of the data.

Authentication is the method used to validate that necessary permissions are held to access the communicated data. Each of the reviewed protocols has its own processes for authentication. Bluetooth uses a process called shared secret for its authentication process. Shared secret involves a secret password or passphrase being shared between communicating parties either beforehand or at the start of the communication session. These secret keys are created using a key-agreement protocol such as a public key or the previously discussed symmetric key process. Public key protocol involves data being encrypted with a key that is known to the public and a private key. Data is transferred to the intended receiver who will also have the private key and so be able to decrypt the data using this and the public key.

Networks using the ZigBee protocol use cipher block chaining message authentication code (CBC-MAC) for its authentication process. CBC-MAC is a technique used for a block cipher to construct a message authentication code that consists of a secret key and a short piece of data that will be authenticated. Data is encrypted in blocks. These blocks are chained together. The encryption of each block depends on the encryption of the previous block. If any individual block is changed without using the correct key, it will have unpredictable changes on each subsequent block in the chain.

Networks utilizing Wi-Fi use Wi-Fi Protected Access II (WPA2) for authentication. WPA2 is based on the IEEE 802.11i standard providing a level of security similar to that utilized by government agencies. WPA2 includes both a personal and enterprise version for different levels of security. WPA2 personal uses a password protection for security. WPA2-Enterprise uses the server to verify the network users. Through the processes of encryption decryption and authentication process, there is a slight chance for data becoming altered or corrupted. Steps must be taken to ensure the data is protected and received in its intended format.

Data protection that relates to the reviewed wireless protocols involves a cyclic redundancy check (CRC). A cyclic redundancy check (CRC) is commonly used for error detecting in digital networks as well as storage devices to identify unintended changes to raw data. Data enters the system in blocks that get checked based on a remainder of polynomial division of their contents; upon retrieval, the calculation is repeated and necessary corrective actions are taken on data presumed to be corrupted if the data does not match. Table 1 shows the relationship between the three IEEE standards.

Table 1

Comparison of the Bluetooth, Zigbee, and Wi-Fi Protocols

Standard	**Bluetooth**	**ZigBee**	**Wi-Fi**
IEEE Spec.	802.15.1	802.15.4	802.11a/b/g
Frequency Band	2.4 GHz	868/915 MHz; 2.4 GHz	82.4 GHz; 5 GHz
Max Signal Rate	1 Mb/s	250 Kb/s	54 Mb/s
Nominal Range	10m	10-100 m	100 m
Nominal TX Power	0-10 dBm	(-25)-0 dBm	15 - 20 dBm
Channel Bandwidth	1 MHz	0.3/0.6 MHz; 2 MHz	22 MHz
Coexistence Mechanism	Adaptive freq. hopping	Dynamic Freq. Selection	Dynamic freq. selection, transmit power control (802.11h)

Table 1 (cont.)

Comparison of the Bluetooth, Zigbee, and Wi-Fi Protocols

Basic Cell	Piconet	Star	BSS
Extension of Basic Cell	Scatternet	Cluster Tree, Mesh	ESS
Max number of Cell Nodes	8	>65000	2007
Encryption	E0 Stream Cipher	AES block cipher	stream cipher , AES block cipher
Authentication	Shared Secret	CBC-MAC	WPA2(802.11i)
Data Protection	16-bit CRC	16-bit CRC	32-bit CRC

Upon reviewing the data in Table 1, some comparisons can be drawn between the different technologies. If the network is going to be spread out across a large area, it would be best to set up either a ZigBee or Wi-Fi network. In the case of wide area networks where cost is not an issue, ZigBee would be the best choice because of the number of cells that are able to be linked together. If larger amounts of data are going to be passed, then Wi-Fi may be the best option. If encryption is a key issue, Wi-Fi is the better choice as one has the option of stream cipher or the AES block cipher techniques. After comparing all of the characteristics of the three protocols, if cost is not an issue, Wi-Fi would be the best option for use in a structural health monitoring system as it provides the ability to send larger amounts of data at a faster rate across a larger network with a choice of security. If cost is an issue, devices that operate under ZigBee would serve the needs of most applications. The only down side to these devices is the lower maximum signal rate and band width, reducing the amount of data communicated.

2.2.3 Wireless motes and sensors. Wireless sensors, as the name would imply, are battery powered sensors that are able to communicate wirelessly via radio, Bluetooth, wireless gateways or any number of other minimally wired or completely untethered means of data transfer, needing no physical connection to neighboring sensors or computers to communicate data. The sensors can vary widely in size, shape and design based on their functionality.

Wireless motes are devices that allow for the combination of multiple sensors. Some of the wireless motes are inclusive, having multiple sensors hardwired into them directly. Other motes, such as the Imote2, are stackable. Stackable motes allow for much more customization as the user is able to connect multiple sensors together to fit the needs of the project and location in which the motes will be placed.

For the purposes of this research, the Imote2, Cricket and SunSPOT motes were all tested. The SunSPOT is a relatively small, wireless, battery powered sensor mote developed by Sun Microsystems to be utilized in a wireless sensor network (WSN) [01]. SunSPOTs are embedded with microprocessors that run Java software. This allows programmers to more easily create unique projects that can be customized to meet the needs of developers and their given deployment environments.

The Cricket motes are primarily utilized as an indoor location system. They allow for widespread and sensor-based computing environments. Crickets use a combination of RF and ultrasound technologies to communicate location information across a network. This network is made up of beacons and listeners. Beacons are mounted on walls and ceilings of a structure to be monitored and publish data gathered from its designated monitoring area to the RF channels using concurrent ultrasonic pulses. Listeners continuously listen to RF signals and, once the first bits of data have been recognized, continue listening for all corresponding ultrasonic pulses. The

Cricket motes are programmed in nesC, a specialized version of C that runs on the TinyOS environment, an event-driven OS for devices that have restricted resources.

The Imote2 is also programmed using nesC. The Imote2, developed by Intel, gives users a cost-effective platform for developing and evaluating wireless sensor networks as well as associated applications [13]. The motes were created using the low power PXA271 XScale CPU with variable processing speeds for optimizing power consumption. The imote2 also includes a ChipCon 2420 802.15.4 compliant radio with an internal antenna that allows for data to be passed up to 30 meters [14]. For applications that require data to be passed over longer distances, an external antenna can be attached as well to increase the transfer distance from 30 to 100 meters.

Imote2s do not possess inherent sensing capabilities, but instead they provide a flexible medium for a range of sensing applications by allowing various stackable sensors to be attached to address the needs of various systems. Imote2s do not have onboard analog-to-digital converters (ADC) and as such the mote is only compatible with digital sensor output. Even with this limitation, the Imote2 has a wide variety of compatible stackable sensors that allow the Imote2 to be an ideal device for many different wireless sensor network applications.

There are several parameters by which motes can be compared. CPU (central processing unit), radio, power consumption and needs and physical characteristics all come into play when choosing the best device to suit the needs of an implementation. In some cases, a negative characteristic may need to be overlooked in order to gain access to another trait that is necessary for the overall cohesive functionality of the system.

When comparing the CPUs of different motes, there are a few key factors that are usually taken into account: processor type, amount of included memory and clock speed. These factors

play a pivotal role in the functionality and capability of the motes. The amount of memory a device has dictates the size limits on applications that are able to be loaded onto that device. SRAM (static random-access memory) is actually a semiconductor that utilizes a form of circuitry called bistable latching to store each bit in a volatile form or memory storage, which means the data will be lost once the device powers down. Another form of memory, flash memory, is non-volatile memory. This means that a device that utilizes flash memory to store data will retain its memory after the device has been powered off. These two memory types as well as others are compared against the needs of the system and the foreseeable applications that will be created and loaded on the devices to discover the best avenue for data storage and transfer. Closely relate to the size of the memory is the word size. The word size is a unit that is used to convey the size of the instructions that a device is able to process. Another key aspect related to the CPU is the clock speed or clock rate. This refers to the frequency at which the CPU is running. The clock speed limits the rate of transfer and processing for data internally processed.

The next key characteristic of a wireless mote, and probably one of the first that comes to mind for most people researching them, is the traits of the mote's radio. The features of a mote's radio are chiefly based on two things, the type of transceiver and the type of antenna incorporated in the mote's design. The transceiver is comprised of a transmitter and a receiver for sending and receiving packets of data. The transceiver plays a large part in determining the frequency band as well as the data transfer rate of the mote. The frequency band and data transfer rate determine the number of bits per second that can be sent and received to and from the device. The distance at which those packets of data are able to be successfully sent is determined largely by the type of antenna incorporated. Some motes may use an internal

antenna, external antenna or a combination of the two. Usually a mote that has an external antenna, either included or as an added physical feature, is able to send and receive data successfully over a greater distance.

Another issue of concern regarding the selection of a mote is the power needs and consumption. Motes utilize different amounts of power based on the mode in which they are currently. In many sensor networks, the majority of motes spend most of their time in sleep mode. In sleep mode, the motes utilize considerably less power. In addition to the amount of power utilized is the type of power source employed, whether it is the standard AA battery used for many portable devices, the smaller AAA batteries that can be used in many of the same device types but have less weight, or some form of rechargeable battery such as the lithium ion battery.

Table 2

Comparison of the Imote2, Cricket and Sunspot Devices

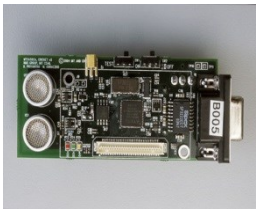
	IMote2	**Cricket**	**SunSPOT**
			
CPU			
Processor	Intel PXA271	Atmega128L	Atmel AT91RM9200
Clock Speed	14-416MHz	7.373MHz	180MHz
Memory			

Table 2 (cont.)

Comparison of the Imote2, Cricket and Sunspot Devices

SRAM Memory	256 kB +32M external	4kB	512kB
FLASH Memory	32MB	128k	4MB
Word Size	32 bits	8 bits	32 bits
Radio			
Transceiver	TI CC2420	TI CC1000	TI CC2420
Frequency Band (ISM)	2400.0 – 2483.5 MHz	2400.0 – 2483.5 MHz	2400.0 – 2483.5 MHz
Data Rate	250 kb/s	250 kb/s	250 kb/s
Range (line of sight)	~30 m With integrated antenna 100m with external antenna	20-30m	80 m
Power			
Current Draw In Deep Sleep	390µA	< 15µA	30 µA
Mode			
Current Draw In Active Mode	66mA	8mA	80mA
Battery Board	3x AAA	2XAA	Lithium Ion Battery

Table 2 (cont.)

Comparison of the Imote2, Cricket and Sunspot Devices

Battery Voltage	3.2V – 4.5 V	2.7V - 3.3V	3.7V
Mechanical			
Dimensions	36mm x 48mm x 9mm	58mm x 32mm x 7mm	63mm x 38mm x 25.4mm
Weight	12g	18g	33.49g

Finally, the physical characteristics (size and shape) of the mote have to be taken into account. Depending on the needs of the system, the size and shape can play a huge role. Many different system architectures have different needs. In the case of a structural health monitoring system in an airplane, in addition to the size, the weight is a critical factor. Though the weight of the individual motes is not very significant, the weight of a network of motes added to a plane may have a major effect on the functionality of the plane while in flight. Table 2 compares the characteristics of three motes researched for possible us in the later discussed sensor architecture

Upon reviewing the three different devices, it is clear that each has its own strengths and weaknesses. The SunSPOT has the least amount of learning curve as it was created with student research in mind and applications can be programmed using Java. It also comes equipped with a rechargeable lithium ion battery, which eliminates the financial burden of replacing batteries. On the negative side, the SunSPOT is inflexible in the data that it can sense as the device has a closed system design. The features of the SunSPOT mote make it perfect for users who are new to the world of wireless sensors and are seeking to acclimate themselves to the techniques involved in data capture applications.

The positive aspects of the Cricket mote are its flexibility and its power use. The Cricket motes are able to utilize a list of attachable sensors for flexibility of data capture. Though two AA batteries are needed to power the device, it uses only 8mA in active mode and less than 15μA in sleep mode.

The Intel Imote2 devices are an upgrade to the Cricket devices. The benefits to the Imote2 are its ability to transfer larger amounts of data than the other two devices; it is stackable and can potentially communicate data across a much larger area. The Imote2 has a clock speed of up to 416MHz, which means that it will be able to communicate larger packets of data at a faster rate. Similar to the Cricket, the Imote2 is able to stack multiple sensors on top of each other to accommodate the needs of a wide range of sensor network. With an external antenna attached, the Imote2 is able to transfer data up to 100 meters away, allowing for fewer devices in networks where sensing nodes are spread far apart. Though the Imote2 has a much higher learning curve to program, of the three devices reviewed, it is the best all-around device for data capture within a wireless network because of its flexibility, included memory, clock speed and range of data transfer, all of which are key issues when deciding what technology to incorporate in a wireless sensor network.

2.2.4 TinyOS/nesC. TinyOS is a light-weight, component based operating system written in nesC [30]. It is primarily utilized for wireless sensor networks. TinyOS started out as a collaborative effort between researchers at University of California, Berkeley, Intel Research and Crossbow Technology. It was designed to be an event-driven operating system for sensor network motes and other devices with limited resources.

The one aspect of nesC that differs the most from C is its linking model. The primary challenge and intricacy is not in writing software components as much as it is in combining a set

of components into a concise working application [31]. nesC allows for construction and composition to be separated. This allows small applications are built using *components*. These components are then assembled, or "wired", to form larger, more functional applications. Components utilize tasks as a form of internal concurrency. Control threads are passed back and forth through interfaces to the components. The threads are usually seen either in a task or hardware interrupt.

Component behaviors are specified based on their set of interfaces. Components are able to provide or utilize interfaces. These interfaces represent the functionality that the component gives to the user while the user interfaces represent the functions needed to perform the component's job.

Function sets are designed based on the provider, commands, and the user events. Using this design, a single interface can be used to represent relatively complex interactions between components. This is important because TinyOS commands can be lengthy and are non-blocking; meaning an event is used to signal their completion. Through interface specifications, components are not able to call send commands unless an event is provided for event sendDone to signal the completion of the event.

The hierarchy of command calls is top to bottom. Calls will filter down from the top level of the application down to components that are close enough to the hardware utilized. Conversely, events are sent up the chain command. Some events are bound to hardware interrupts.

Interfaces are used to statically link components together. Combining components in this manner allows for increased runtime efficiency while allowing for better static analysis of programs. This also encourages a robust design.

For security and data encryption, both TinySec and MiniSec are utilized in securing sensor network communication. Both TinySec and MiniSec work in conjunction with versions of TinyOS, TinySec being used in the TinyOS 1.x environment and miniSec being utilized in the TinyOS 2.x environment [30]. The basic idea behind both security protocols is that the message is encrypted using a key, then passed over the network and decrypted on the other end by the receiver. The key is set to a default value but can be changed by the user to fit a specific need. This allows for data to be passed back and forth across the network much more securely, minimizing the chances of data leakage throughout the network.

2.3 Agents

An *agent* can be defined as an autonomous, problem-solving, computational entity that is capable of effectively processing data and functioning singularly or in a community within dynamic and open environments. They contain beliefs, capabilities, choices and commitments within the context of their given environment. By the usual definition, an agent does not explicitly refer to a piece of software [05]. In some cases, the human(s) interacting with the system can be considered agents as well and so may be included in the blueprint of the multiagent system. For the purposes of our architecture, the term “agent” shall be used to refer to the software agents both deployed directly into the wireless sensor network and or the agents that make up the control hierarchy for the passing, manipulation and storing of data.

There are many benefits of utilizing an agent-based system for control as well as distribution of data. In a well-defined system of agents, agents are able to communicate with each other and the user, propose projects, bid on those proposed projects and even replicate in special cases where they are programmed to do so. Agents are ideal for environments where new challenges arise and require flexibility and the ability to breakdown and subcontract tasks.

Regarding intelligence, agents are able to be created with the ability to follow a specified set of instructions continually throughout the life cycle of the system, and they can also be created with the ability to learn and adapt to their environment and gain an actual level of knowledge and understanding about their given working environment. Each avenue offers its own distinct set of benefits and challenges. In the case where the designer decides to go the route of a set list of rules for a perceivable set of circumstances, the encoding process is a much less daunting task as the designer will simply design the agents to respond to foreseeable events. The downside to this is the agents are not as adaptable and so the system may have issues if a critical event occurs that was not previously accounted for by the designer. In the case where it is decided that it is best to encode agents with the ability to learn from their environment and situations as they arise, the system becomes much more flexible and robust. The challenge in this case is instead that the initial programming is slightly more complex. Whether the designer decides to use either option or a combination of the two will be predicated upon the needs of the overall system. Once a decision about intelligence of the agents is made, the next issue to address is how the agents will interact with each other within the network as well as with the user. This next concern can be succinctly addressed by the capabilities of the Contract Net Protocol.

2.3.1 ACL. An Agent Communication Language (ACL) is the language for communicating *knowledge* between two or more agents regardless of their respective hardware platforms, operating systems, architectures, programming languages or representation and reasoning systems [04]. There are a number of proprietary and open-source ACLs currently being utilized. Because of this, agents from different systems are not able to communicate with each other unless they use the same ACL. The main traditions in defining the semantics of an

ACL are to define the language based on mental attitudes and or social commitments [07]. These languages all have communication operators with requirements and properties and involve agents taking on roles such as sender and receiver.

In the early 1990's, organizations such as DARPA (Defense Advanced Research Projects Agency) funded projects for developing protocols for exchanging knowledge between autonomous information systems [09]. Two of the most utilized ACLs are the KQML (Knowledge Query and Manipulation Language) and the ACL developed by FIPA (Foundation for Intelligent Physical Agents). Both of these languages have been proposed as standards for agent communication.

KQML is an older ACL than FIPA-ACL (see below) and currently it is somewhat out of date [09]. The KQML agent communication language is a message-based language. Message-based refers to KQML defining a common format for messages. These messages are similar to some of the common components of OOP (Object-Oriented Programming). Messages have *performatives*, which are similar to the concept of classes in OOP. KQML messages also contain *parameters*. These parameters are similar to attributes or instance variables in OOP. The main KQML parameters are included in the table below. Additional parameters have been added to this list over time to better handle a wide variety of messages.

Table 3

KQML Performatives

Performative	**Description**
: content	Content of the message
: force	If the sender of the message will ever deny the message content

Table 3 (cont.)

KQML Performatives

: reply-with	Dictates if the sender is expecting a reply and if so, provides an identifier for the reply
: in-reply-to	Refers to the : reply-with parameter including the dictated identifier.
: sender	Identifier for the sender of a message
: receiver	Identifier for the intended recipient of a message

Agents using KQML to communicate can be implemented with different programming languages and paradigms. This flexibility allows information that agents have to be represented internally in many different ways. Though some agents utilizing KQML may not have an internal representation of the knowledge in question, for communication purposes, agents treat other agents as if they do contain some internal representation of that knowledge. This internal knowledge, whether present of not, is known as a *virtual knowledge base*.

FIPA-ACL was developed in 1995 and, just like KQML, is designed to work with any number of content languages and ontologies [08]. FIPA-ACL is the standard ACL for multi agent systems endorsed by FIPA (Foundation for Intelligent Physical Agents). FIPA is an IEEE organization founded in 1996 that promotes the advancement of agent-based technology and the interoperability among multiagent systems and other technologies [11]. Both FIPA-ACL and KQML define an outer language for messages. Also, both ACLs define performatives such as *inform* for defining intended message interpretations. Performatives are words or phrases that both describe what they do while performing that intended act such as “I now pronounce you

man and wife." FIPA-ACL and KQML also have similar syntax for messages [09]. The structure of a message sent using FIPA-ACL and that of KQML are similar in both structure and attribute fields. The main difference between KQML, as depicted in Table 3, and FIPA–ACL, depicted in Table 4, is the available performatives that can be used for agent communication.

Table 4

FIPA-ACL Performatives

Performative	Description
: accept-proposal	Used when an agent accepts a proposal made by another agent
: agree	Used by one agent to state that it agrees to a request made by another agent and indicates that the agreeing agent intends to carry out the request.
: cancel	Used by a requesting agent to inform other agents that it no longer needs a particular action to be carried out.
: cfp (call for proposal)	A *task-sharing* performative used to initiate negotiation between two or more agents. This message would include both an action needing to be performed and a condition under which that action should be performed.
: confirm	Allows the sender of a message to confirm the truth of a message to a recipient that the sender believes may be unsure about the truth of the content
: disconfirm	Used to indicate to a recipient that the sender believes the content of a message is false
: failure	Allows an agent to inform another agent that an action attempted to be performed failed.
: inform	Used as the basic mechanism for communication. The basic idea is that the sender of an inform message wants the recipient of the inform message to believe something.
: inform-if	Used as a true false inform message requesting that a recipient of the message inform the sender if the content of the message is true or false
: inform-ref	Used when the sender of a message is requesting the value of a message from the recipient of the message.
: not-understood	Used by a recipient agent of a performative request to state that the action has been performed but the agent who performed the action is unclear as to why the action was performed.
: propagate	This message consists of two things: another message, and an expression that lists a set of agents. The recipient of the propagate message will ideally send the embedded message to the agent(s) listed

Table 4 (cont.)

FIPA-ACL Performatives

	in the message
: propose	Allows an agent to make a proposal to another agent.
: proxy	This allows sender of a message to treat the recipient of a message to
	act as a proxy for a set of agents.
: query-if	Used to allow a sender to request if a specific statement, included in the message content, is true.
: query-ref	Used by a sending agent to request the value of an expression
: refuse	Used for one agent to tell another agent that it will not be able to process a request
: reject-proposal	Used for an agent to reject a proposal that was made as a part of a negotiation.
: request	This performative allows an agent to request another agent to perform an action
: request-when	This is used when a sending agent wants an action to be performed when a specific statement is true.
: request-whenever	This is used when a sending agent wants an action to be performed whenever a specific statement is true.
: subscribe	An agent uses this to request that any information regarding a statement be sent when something relating to it changes.

In addition to the performatives, there are several other differences betweek FIPA-ACL and KQML [08]. In the FIPA-ACLsemantic model, agents are not allowed to manipulate another agents virtual knowledge base directly. FIPA ‘s architecture also includes an agent management system specification that specifies services for managing agent communities. Another difference between FIPA-ACL and KQML is the semantics. The semantics of KQML has terms to define *preconditions*, *postconditions* and *completion conditions*. Preconditions indicate the states necessary for an agent to send performatives and for the receiver of that performative to accept and successfully process it. Postconditions are used to describe the states of the receiver after a performative message has been successfully sent from a sender and then received and processed by a receiver. Completion conditions specify the final state once a conversation has taken place. The completion condition indicates when a specified performative has been fulfilled.

The semantic language of FIPA-ACL messages can represent objects, propositions and actions. The semantics of each speech act is specified with semantic language statements that describe the *feasibility pre-conditions* and *rational effect.* Feasibility pre-conditions layout conditions that are necessary for the sender of the message. Rational effects describe the effect that an agent will experience as a result of performing the act as well as specifying the conditions that should hold true of the recipient. Though both FIPA-ACL and KQML are able to function as agent communication languages, the system architecture discussed below assumes FIPA-ACL for agent communication as it is the current standard for agent communiction and is used by JADE.

2.3.2 Multiagent systems. A multiagent system can be defined as a collection of agents, autonomous in nature, all working within a system to achieve individual as well as group goals for the betterment of the system and the maintenance of its continued functionality. Examples of multiagent systems, relating to the general definition of agents, in which individual members autonomously work together to break down large tasks in to smaller subtasks and accomplish their goals through communication and cooperation, appear throughout nature. One example of this would be the ant which functions as an autonomous member of a cohesive community. It appears in looking at some of the more social insect colonies such as the bee or the ant. It also appears frequently in our own society in situations such as the workplace where each employee has their own role but may utilize the help of other employees in able to solve a problem that may not have been solvable on its own. Breaking down and assigning tasks based on individual capabilities and the good of the whole allows both the ant colony and the fortune 500 company to run smoothly and adapt to unforeseen situations that could otherwise disrupt the flow and overall structure of the whole organization.

According to Tim Wooldridge, author of the leading text on multiagent systems, *An Introduction to MultiAgent Systems*, multiagent systems (MASs) address the five main trends that have driven the advances in computing: ubiquity, interconnection, intelligence, delegation and human-orientation [09]. The first, ubiquity, is made apparent in the ability of a system to span variable landscapes. Some networks can be found in a single classroom while others may connect across countries. This is achievable chiefly due to two things, the type of hardware being used in the system and the software or programming language used to create the agents. Based on the sensor, its internal or external antenna and its accessibility to the web, data may be passed anywhere from a few feet at a time to meters at a time. The difference the type of software makes when creating agents can be seen when comparing a software package such as JADE with, say, Agilla, where JADE is made for agents to be connected directly to workstations and traverse a hardwired network and, in contrast, Agilla is made for mobile agents being utilized on sensor nodes and moving or "hopping" from one sensor node to another to acquire and pass information throughout the network. *Figure 5* illustrates the second trend, interconnection. The agent organizational structure depicted in *Figure 5* is a scenario where agents in two multiagent systems are working side by side to complete their respective tasks, with one agent overlapping between the two systems for communication.

In this depicted setup, three of the agents have access to the environment, and these agents are able to communicate data and requests with other agents in their respective multiagent systems. This setup allows for the agents in the depicted architecture to communicate data between each other as well as to recipients outside of their respective environments.

The next trend is intelligence. One main requirement for an agent is autonomy, or the ability to make decisions for itself based on the task it is designed to perform. With the addition

of a rule engine such as Jess, later discussed, agents are able to make decisions regarding a broad array of situations related to their tasks.

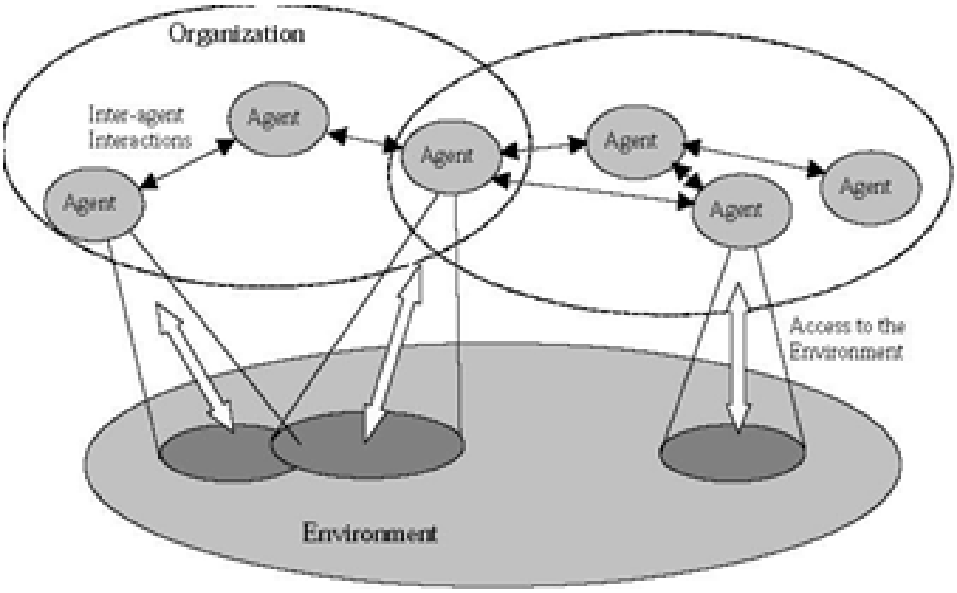


Figure 5. Overlapping Multiagent Systems [09].

The next trend is delegation. This mainly appears in the cooperation and bargaining process. The later discussed Contract Net Protocol allows agents to break down large tasks into smaller subtasks and delegate those tasks to other agents. Agents are able to bid on subtasks based on their respective capabilities. This allows agents to cooperate and bargain to create a situation where all parties have some benefit from the relationship.

The last trend is human-orientation, which relates to concepts and metaphors of the human-computer interaction moving away from machine-oriented and closer to the way that we ourselves view and understand the world. In terms of the MAS, the trend to human-orientation is evident in the agent communication structure as well as agent behavior models. Agents by definition are autonomous entities that are able to communicate with each other, break down large tasks into smaller tasks for ease of processing and interact with their environment. It is also possible for agents to have beliefs, desires, intentions, and a sense of understanding about their

fellow agents and the environment in which they exist, all of which we as humans are able to do naturally with minimal effort.

2.3.2 Contract Net Protocol. The Contract Net Protocol is a high level protocol for achieving efficient cooperation through task sharing in networks of communicating problem solvers [10]. As the name implies, the main concept used in the Contract Net Protocol is contracting. The basis of this was inspired from the way companies organize and process contracts put out to tender.

As previously discussed, a multiagent system is made up of autonomous agents working together to decompose large problems into smaller sub-problems that individual agents are able to process. Each individual agent in the system is given a set of tasks, rules and available actions in order to accomplish its goals. At times, individual agents are unable to accomplish larger goals by themselves either due to the unavailability of resources or because they are unable to process some required prerequisite tasks. It is during these situations that the agent must call out and ask for help. The agent in need of assistance advertises a task to other interested agents and then acts as the manager of that task.

The Contract Net Protocol is a hierarchical data network for subcontracting tasks within the multi agent system and breaking down tasks into subtasks that can be more easily performed by a group of agents than by an individual agent. Figure 6 is a block diagram illustrating the interaction between agents in a Contract Net Interaction Protocol utilizing FIPA-ACL [04]. The figure depicts the initiator of a call for proposals sending a request to *m* number of participant agents. Uninterested agents or agents unable to process the request within the proposed deadline will refuse the request. Agents that are interested respond with a proposal. The initiating agent will review the proposal and reject all but the one it feels best fits the task's criteria. The

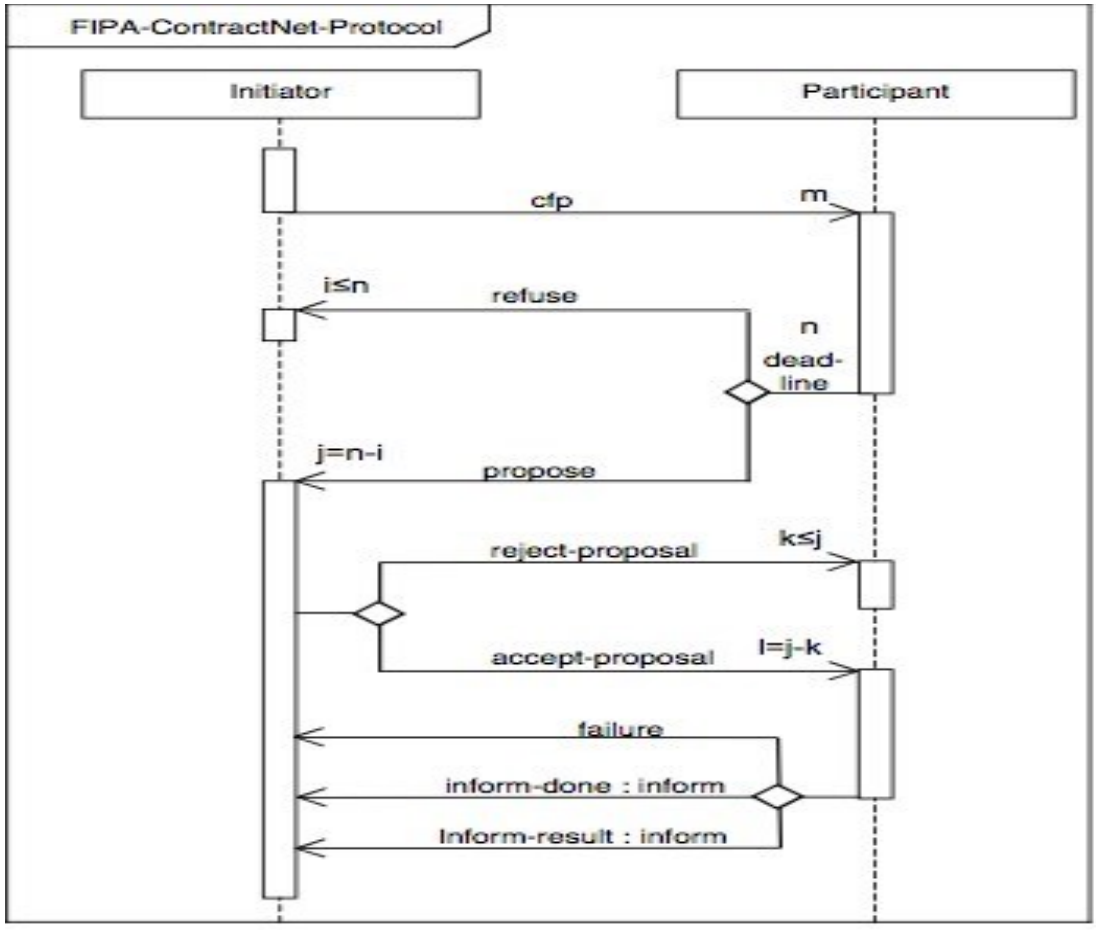


Figure 6. FIPA Contract Net Interaction Protocol [04].

initiating agent will send the agent with the winning proposal an acceptance message. Finally, the remaining participant will attempt to process the request and will send a message to the initiator stating whether the task was done or they failed and also any relative results of that request. Winning bids can be based on any number of criteria, from the time to process the requested task to possible future assistance on an issue that will benefit the bidder in accomplishing its own task. Through this process of communication and cooperation, large tasks can be contracted and subcontracted to appropriate agents, which creates a more efficient and effective use of resources and data transfer. This also allows for a hierarchy to naturally develop based in assignment of tasks and subtasks.

2.3.3 Gaia. In order to reasonably guarantee that a system comprised of multiple autonomous agents is able to act and operate efficiently and with minimum error, a large amount of initial planning must go into the overall design of each individual agent. We must understand the purpose, function, movements, interactions and possible detriments of each agent involved. Utilizing software engineering methodologies, we are able to systematically analyze and design our system, allowing us to create an efficient and capable system. For this purpose we turn to the Gaia methodology.

Gaia is a methodology created specifically for the analysis and design of agent-oriented systems [09]. It allows the designer to create a schema of each agent that depicts its individual actions and processes as well as the interactions that occur between agents. Schemas are akin to blueprints utilized by architects, but, in this case, they give the software designer and implementer an overall picture of the multiagent system being created as opposed to the building being constructed. The terms of an organized society of multiple agents can be defined, depicting the roles and interactions of the agents according to predefined protocols created for each class of agent.

Utilizing Gaia involves two main development phases, analysis and design. A depiction of the full Gaia process is shown in Figure 7. In this figure, components and models that are needed during the analysis and design phases are shown. The process begins with the analysis phase, which chiefly is a process of collecting and organizing the specifications of the system.

One initially articulates a *role model,* or a model of the roles each agent type will take as well as their abilities, and an *interaction model*, which is a model of how agents will interact with each other. The role model and interaction model are reviewed based on the perceived environment in with the system will be implemented.

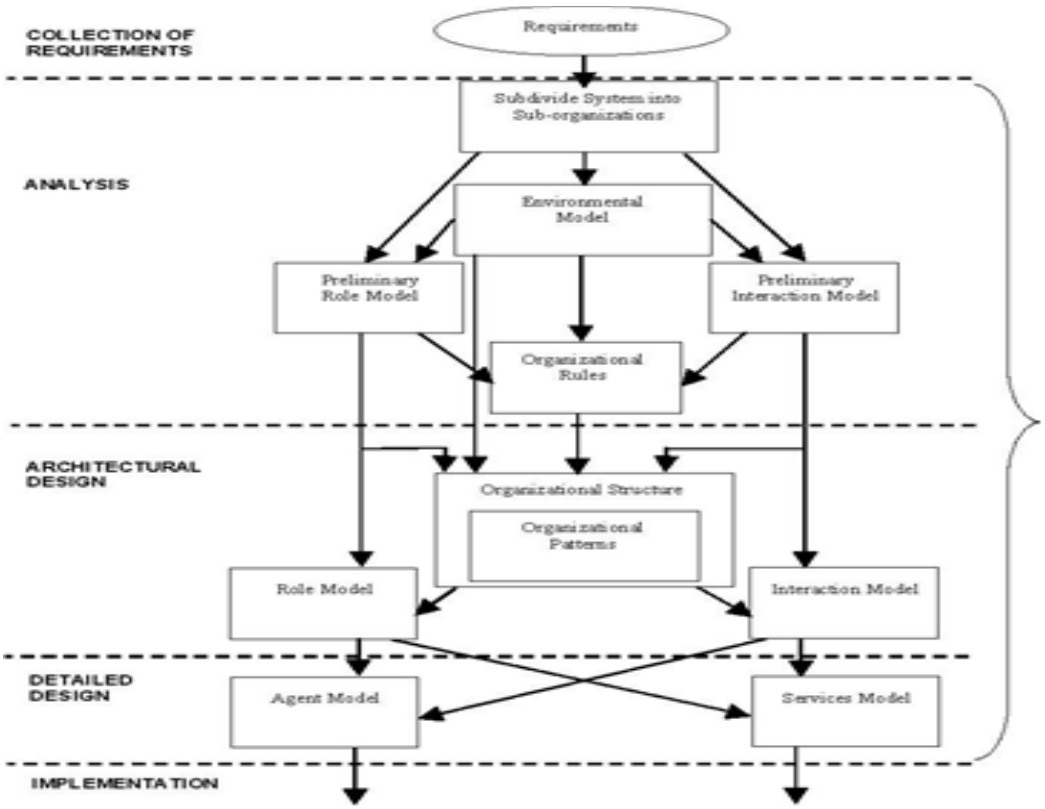


Figure 7. Complete Analysis and Design Model [43].

This allows for the beginnings of the rules that will govern the system. During this phase, the designer attempts to answer the following:

- What are the overall goals of the organizations and their expected global behavior?
- What is the perceived environment model?
- What will be the preliminary interaction model?
- What will be the rules that the organization should respect and enforce in its global behavior?

From these question, the design phase attempts to create a layout of the following:

- The overall system organizational structure in terms of its topology and control regime.
- The completion of the interaction model, which shows how the different agents will

interact with each other and the preliminary roles that dictate the functionality of each agent type. This section breaks down the individual agent capabilities and assigned tasks. The role model also illustrates the steps which they take to process those tasks.

2.3.4 Software. There are several agent based software packages available for use in creating a multi agent based system. Each software package has its own set of strengths and weaknesses. For the purpose of this implementation, we chose to use JADE as the main programming package for creating our multiagent system as well as the Contract Net Protocol by which they will interact. This is mainly due to JADE being a Java based Development framework for creating agents. For the rule engine which the JADE agents will follow we chose Jess. Jess is a rule based system also implemented in Java which makes it a perfect medium for encoding the rules and regulations which dictate the agent's actions and reactions to a given situation. In addition to JADE, we also chose to use Agilla mobile agent middleware for its ability to create and deploy mobile agents onto a network of sensors. Agilla agents, once deployed, are able to migrate their code, and current state from one sensor to another. This ability helps to regulate the amount of data travelling back and forth between the wireless sensors and the base station.

2.3.4.1 JADE. JADE stands for JAVA Agent Development Framework and as its name would dictate, is a framework for the development of agents [44]. JADE is implemented in the JAVA programming language and works on versions of the Java runtime environment 1.4 or higher. Founded in 1996, FIPA was originally created as a Swiss-based organization that was tasked with creating protocols for developing heterogeneous and interacting agents and agent based systems [22]. Using a middleware that complies with FIPA standards, JADE simplifies the creation of multiagent systems. JADE includes a set of graphical tools that help the user in

both the debugging and deployment stage of the system.

Because the agent platform is able to be distributed across multiple machines, not necessarily even running on the same operating system, JADE multiagent systems are both flexible and robust. The configuration of the system is able to be controlled using a remote graphical user interface. This adds even more flexibility to the design and implementation of the system. If needed, agents are even capable of reconfiguring their own network and relocating their code from one machine to another, provided the machines are connected through some form of network.

Agents created in JADE are able to work collaboratively within a system of their peers. Agents are able to process directives proactively based on rules implemented by the designer. They are also able to communicate and negotiate with other agents as well as directly with the user, based on the given situation, in order to accomplish their goals and assigned tasks more efficiently. JADE agents are also able to coordinate in order to solve complex problems in a more distributed way so that large problems may be broken down in to more manageable tasks and processed more simply through the cooperation of agents in the network. Environments where the of implementation of a JADE system could potentially be a benefit range from corporate use to day to day tasks, Internet services to mobile environments.

JADE attempts to ease the burden of development for the creator by offering an extensive set of application programming interfaces, or API's, in conjunction with a respectable suite of programming and testing tools. This makes the job of developing intuitive peer-to-peer agent based applications a much less daunting task.

2.3.4.2 Jess. Jess, the Java Expert System Shell, is a Java based system for creating application rules. The software can work independently or, as in the case of the implementation

discussed later, it can work in conjunction with JADE to create a robust and functional network of agents that are able to adapt and respond to a wide variety of situations [23].

Rules can be defined as a type of instructional commands that are applicable in certain situations. We are taught to obey or at least acknowledge rules at an early age. "No running with scissors", "Brush your teeth before you go to bed at night", "Whatever can go wrong will go wrong" and so on. Statements like these could be encoded as rules in simple *if-then* format to be represented as code.

IF

I have Scissors

THEN

I cannot run

END

or

IF

It is night time

AND

You are about to go to sleep

THEN

Brush your teeth

END

The architecture of a typical rule-based system consists of three main components; the *inference* (or *rule*) *engine*, the *rule base,* and the *working memory*. The inference engine controls the process of applying rules to the working memory in order to obtain the system outputs, and

thus a rule-based- system uses the rule engine to derive a conclusion from given premises. Of course, for our use, we are not creating rules for humans to follow but for agents in a multiagent system. Rule engines do not contain any rules until they are programmed in. The rule engine knows how to follow rules without containing any specified information or knowledge as to the individual rules themselves.

The rule engine in Jess utilizes an enriched form of a widely used algorithm called Rete to match the working memory against the rules. This Rete network is the basic structure of Jess's working memory. Utilizing the Rete algorithm allows Jess to trade space for speed. In order to implement the Rete algorithm, a network of interconnected nodes is built. Each node represents one or more of the parts of the antecedent of a rule. Each node contains either one or two inputs to any number of outputs. When facts need to be added to or removed from working memory, they are processed by this network of nodes. Input nodes are located at the top of the network while output nodes are at the bottom.

The inference engine works in cycles that involve the use of a *pattern matcher*, for deciding which rules to apply in a given situation based on the current state of the system, contents in working memory and the *agenda*, which stores a list of rules that could potentially fire. Conflict strategies are used to determine which stored rules out of the ones that apply have the highest priority and as such should fire first, and the *execution engine* actually activates the rules. Next, the rule base contains a listing of all of the rules known to the system. It may also rearrange the rule premises or conclusions in order to make it more efficient or to better clarify the meaning for a more automatic execution. Finally, the working memory contains the data or information the rule-based system is working with. It also holds the premises and conclusions of the rules. similar in design to a relational database to make searching fast and functional.

Jess rule systems have been utilized in a wide variety of commercial software that span the gamut from expert systems used to evaluate mortgage applications and insurance claims to agents that can predict stock prices and buy and sell securities, from intelligent e-commerce sites to videogames.

Jess may be programmed in two different but overlapping ways. The first method is using Jess as a rule engine. Programmers can create and store any number of rules which pertain to a given context or situation and Jess will automatically and continually apply these rules to the data to which it is provided access. These rules usually represent a heuristic knowledge provided by or based on a human expert in a given field.

Jess does not necessarily have to be used as a rule engine. Jess can also be utilized as a general-purpose programming language, able to directly access all of the preexisting Java classes and libraries. This allows Jess to also be used as a dynamic scripting language where applications may need to be deployed in a rapid and changing environment. In contrast to Java's compile-then-run method, Jess is able to interpret code and execute it immediately as soon as it is typed. This allows the programmer to create and test applications interactively and incrementally.

2.3.4.3 Agilla. Agilla is used as a middleware for wireless sensor networks and allows for the creation of mobile agent programs [18]. Agents created in Agilla can proactively migrate their code and state from node to node within the network. This aspect allows the network to be more flexible as the agents themselves decide how best to move and spread out within the wireless sensor network. Agilla agents position themselves in areas that are of high relevance to the integrity of the network as well as the structure being monitored. Because of their position and sharing of available resources, Agilla agents also allow for an increase in the utility of the

wireless sensor network as a whole because they allow for a system where the processing is taken to the data rather than the data being taken to the processing. Another key benefit is that Agilla agents can be injected directly into a pre-existing network. This enables the network to be re-tasked and optimized in an efficient and seamless manner.

There are many benefits to incorporating mobile agents into a system. Mobile agents reduce the network load. Reducing the network load is possible due to the fact that the agents are able to reduce the flow of raw data across the network [19]. Mobile agents are able to go to the nodes and transport pertinent data instead of the entire load of raw data passing from the sensors directly to a base station. Mobile agents help to overcome network latency. In responding to real-time events, a controller must adapt and make decisions to best address changes in its environment. This allows for the possibility of reducing latency in response time. Mobile agents can be dispatched centrally from a controller to specified locales in order to execute the directives of the controller directly. Mobile agents are able to encapsulate protocols [20]. If the protocol needs to be upgraded in the future, only the mobile agent needs to be updated. Mobile agents execute asynchronously and autonomously. This means that a mobile agent can execute its commands without the direct aid or connection to a home machine. If needed, the home machine can connect to the network later to retrieve the mobile agent and its data. Mobile agents are able to adapt dynamically to different situations. They are robust and fault-tolerant. This is due to their ability to migrate. If a host machine is being powered down, a notification could be sent to the mobile agents, allowing them to migrate to another machine and continue their tasks.

2.4 Data Fusion

Data fusion can be defined as the use of techniques to identify and combine different sources of data streams into actionable, discreet items for use in achieving inferences that will be

more custom-tailored to a specific purpose or use in order to better achieve a predetermined goal [15]. The process is utilized in order to filter through and quantify large amounts of data in order to gain a level of understanding so that later a diagnosis can be made about the current state of a system as well as a prognosis about the foreseeable future.

Based on the standards set by the Joint Directors of Laboratories (JDL) model, there are six levels of processing data, Level 0 to Level 5 [16]. The five levels of the JDL Model are depicted in Figure *8*.

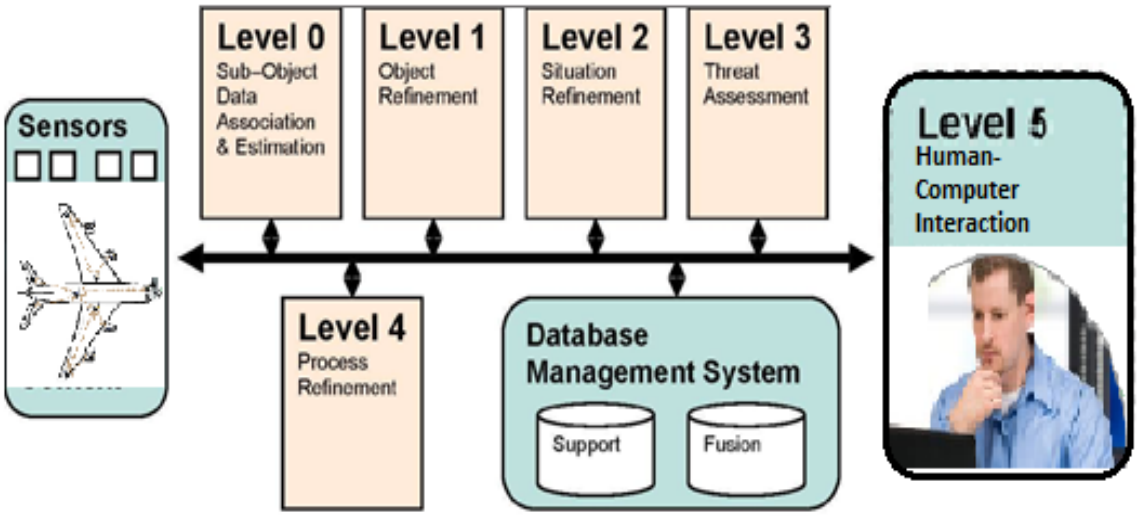


Figure 8. Data Fusion Model.

Each level seeks to organize the data and gain a better understanding so that, in the next level, a more complete picture of the current situation can be gained. Figure 8 illustrates the levels in the JDL model starting from the sensor to the human computer interaction.

Level 0 of the JDL model relates to the refinement and preprocessing of the data from the sensors. Preprocessing of the acoustic emission signals would involve extrapolating feature vectors from the sinusoidal waves that are produced. Once preprocessing has completed, the results are made available to other levels of the JDL model.

Level 1 of the JDL model, object refinement, relates to refining the preprocessed data to arrive at useful classifications. Object refinement would classify the feature vectors from Level 0 as originating from specific kinds of acoustic emission events.

Once the classifications have been made, Level 2 of the JDL model, situation refinement, outlines the total situation by reviewing and quantifying multiple classifications. Al Level 2, the system will produce a diagnosis of the current state of the structural health. In relation to acoustic emissions, numerous acoustic emission signals are reviewed for patterns and compared against predetermined values to characterize the level of threat and associated level of concern needed to address that possible threat.

Level 3, threat assessment, reviews both the current diagnosis as well as past diagnoses to create a prognosis of the future. In relation to the proposed structural health monitoring system, this stage would be used to develop a picture of the future structural health of the monitored system as well as when the system could potentially fail.

Level 4 of the JDL model, process refinement, relates to reevaluating the whole data fusion process for efficiency and accuracy. This involves reviewing both the software and hardware that makes up the system. Past faults, errors and concerns will be taken into account in order to continually improve the data fusion process.

Finally Level 5, human-computer interaction, relates to the addition of user input and interaction in regards to the efficiency and effectiveness of the system to support human decisions. As experience is gained, issues or possible areas of improvement will be discovered. These issues will be addressed to help continually improve the functionality of the system.

2.4.1 Python packages for scientific computing. Python is an open source, general purpose, high-level, object-oriented programming language which is relatively easy to

understand, extendable and user-friendly. It was designed with the philosophy of code readability [25]. Originally, Python was created to be a teaching language. This means that it was created with the ideas of readability, ease of use, “fits in your head”, incremental sense of accomplishment and deployment in mind. Due to these factors as well as the relative ease of learning and ease of showing introductory computer science concepts, Python has become an industry standard for many computer science applications.

There are several benefits of Python. Programs written in Python are relatively shorter than programs written in many other high-level languages. This is due to a few key reasons. You are able to express complex operations in a single statement using high-level data types. Secondly, python uses indentation in order to group statements. Finally, the user is not required to declare variables and or arguments. Python supports many different programming paradigm, including object oriented programming as well as functional programming.

Another benefit of Python is that it is a cross-platform, object oriented application scripting language. The programmer is able to create script programs, which are small pieces of software that a computer is able to run.

NumPy is a library of classes that adds powerful mathematic functions to the Python development environment [26]. NumPy is the primary Python package for scientific computing. The key points of NumPy are as follows:

- A powerful N-dimensional array object
- A list of sophisticated techniques such as broadcasting [27]
    - Broadcasting in this context is the ability for smaller dimensional arrays to be either extended or replicated in order to work with arrays with larger dimensions.
- A host of tools specialized for integrating C/C++ and Fortran code

- Functions that are useful in linear algebra, Fourier transform, and random number generation

Other than its capable repertoire of functions that allow for scientific calculations, NumPy may also be utilized as an effective multi-dimensional container of generic data. NumPy also grants the user the ability to define arbitrary data-types for custom use and flexibility of coding.

SciPy is another package that adds to Python's functionality [28]. It contains a library of algorithms for mathematics, science and engineering. The array structures in NumPy's library are easily utilized with libraries in SciPy, which provides a host of efficient and user-friendly numerical routines for tasks such as integration and optimization because SciPy is built upon NumPy. Utilizing SciPy, complex mathematic problems can be easily and systematically processed.

2.4.2 Machine Learning. Machine learning is a field that is on the boundary of several different academic disciplines. Principally, it is a combination of computer science, statistics, mathematics and engineering. Applications of machine learning can be utilized in a variety of fields, finance, biology, medicine, physics and chemistry to name a few. In the 1997 book Machine Learning by Tom M. Mitchell, it was stated that "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." [40].

Machine learning is usually studied as a specialized sub-topic under artificial intelligence [40]. Programs learn to recognize patterns in given data and, through either supervised or unsupervised methods, gains the ability to accurately classify those patterns into a set of reasonably reliable taxonomies. It requires a certain amount of statistical and mathematical capability by the implementer in order to truly maximize the potential.

The main purpose of machine learning is the design and development of algorithms that allow computers and programs to dynamically change and evolve over time based on empirical data. The changes result in behaviors that allow computers to better process the gathered data. This data is usually passed through to the program via a sensor, database or directly by the user. One of the main objectives of machine learning is to generalize the given data as well as outcomes in order to learn and update processes for optimum efficiency.

Though some systems attempt to remove the human out of the equation almost entirely, others allow for a more direct human machine interaction. In either case, the human functionality is essential for the base set of intuitive thought processes and characterizations of the data.

The various types of machine learning that can be fit into one of four categories: *supervised learning, unsupervised learning, reinforcement learning and evolutionary learning*. Supervised learning is one of the most common learning types and it involves a training set provided with target responses to a problem. Based on these targets, an algorithm is developed to correctly generalize responses to a problem based on unknown inputs. Unsupervised learning involves no correct response being provided. Instead, the system attempts to identify patterns and similarities between inputs. The system then attempts to categorize these inputs based on the highest level of commonality. Reinforcement learning can be classified as a mixture of both supervised and unsupervised learning. In the reinforcement learning algorithm, the system starts out similar to unsupervised learning by attempting to categorize input by similarities. After the inputs have ben categorized, similar to the supervised learning techniques, reinforcement learning is told whether or not it has categorized the inputs correctly. The fourth class of machine learning algorithm consists of evolutionary learning algorithms. In evolutionary learning

algorithms, learning is based on the way biological organisms learn to adapt to and handle events that arise in an unfamiliar environment based on an idea of fitness, which corresponds to how good the current solution is.

In addition to learning algorithms, there are also algorithms used in machine learning that help to better facilitate the learning process such as preprocessing and reducing data to make it easier for the system to process and categorize. Algorithms in this category would include techniques such as relational perspective mapping and, as in the case of the later discussed system architecture, *principle component analysis* (PCA). PCA involves using eigenvector decomposition of a correlation matrix to transform a set of observations into a set of linearly uncorrelated principle components.

2.5 Acoustic Emissions

Acoustic-emission or AE techniques, as depicted in *Figure 9*, are used to characterize features of sound waves that a material generates when subject to stress to understand changes taking place in the material. When a source crack appears in the material, it sends waves along the surface and into the body of the material in all directions. The affixed piezoelectric sensors detect the waves, which allow their features to be extracted. These transient elastic waves can be detected in frequencies can range from under 1kHz up to as high as 100MHz, though usually the range stays between 1kHz and 1MHz.

There are three main applications for study of AE, which are source location (determining where something occurred), material mechanical performance (the evaluation and characterization of materials and structures), and, finally, health monitoring (which deals with monitoring the health or integrity of a structure such as a bridge or a building). Though the main focus of this project is for overall structural health monitoring of an aircraft, through the process

of monitoring, source location may incidentally addressed as well.

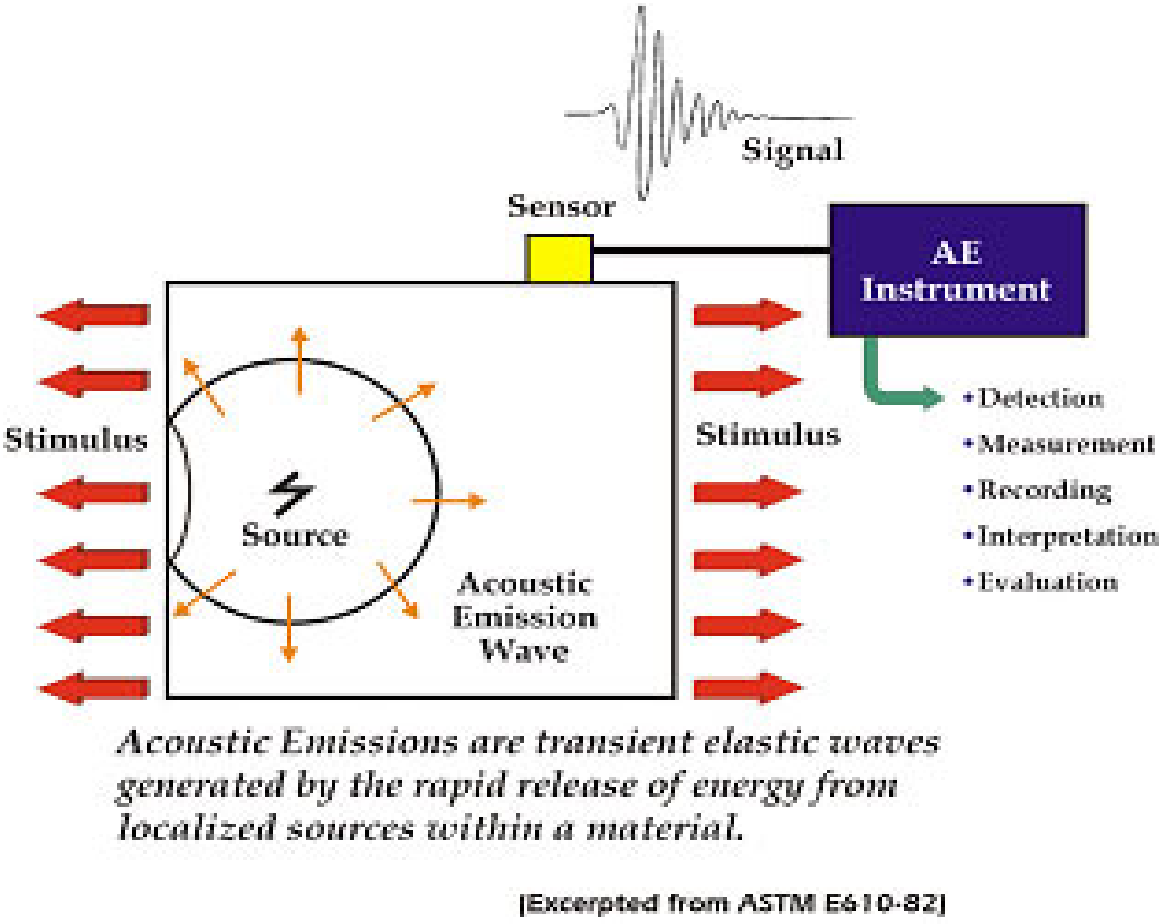


Figure 9. Acoustic Emission [48].

In general, acoustic emission sources that are studied come from material failure, friction (which occurs when two or more surfaces come in contact and rub against each other), or cavitation (the formation of cavities followed by its immediate implosion and finally impact). Testing for acoustic emissions normally takes place between 100kHz and 1MHz. In contrast to ultrasonic testing, the tools used for acoustic emission testing are more reactive. They are designed specifically for monitoring the emissions that are produced within materials in the midst of failure, instead of proactively transferring waves out throughout the material and then reviewing and recording their significance upon their return. Doing this allows for far less power

consumption as the system will process data as it comes in instead of continuously processing data throughout the lifecycle of the structure. Figure 9 is a depiction of an acoustic emission sensor receiving a signal from an acoustic emission event. These sensors allow the system to which they are connected to detect, measure, record, interpret, and evaluate the signals so that the system and the end user can develop an understanding of the event.

CHAPTER 3

Prototype Implementation

This section discusses the implemented prototype structural health monitoring system that utilizes wireless motes controlled by a multiagent system for communicating data and collaboratively achieving tasks. A data stream is communicated between two motes, one at the data source and the other connected to the base station. Data Agents pick up the data and make it available for classifying. Utilizing the Contract Net Protocol, the Monitor Agent assigns selected agents the tasks of extracting features and classifying the event while also allowing requests to be passed down and critical alerts to be passed up.

3.1 Phase1 Implementation

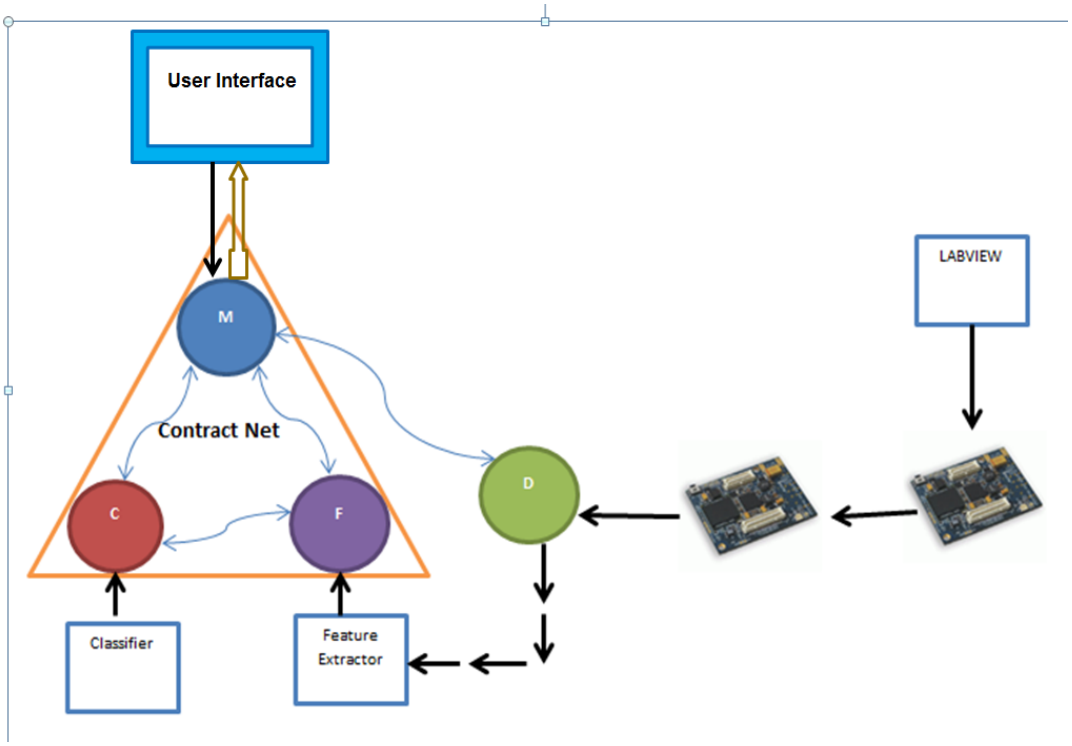


Figure 10. Proposed Phase1 SHM Architecture.

Through research, we have come up with a Phase 1 architecture for structural health monitoring. An illustration of this implemented architecture and its components are shown in

Figure 10. As shown in Figure 10, the system is composed of four agent types: the Monitor Agent, Feature Extraction Agents, Classifier Agents and the Data Agent. Also included in the system are two Imote2 sensor nodes, a LabView application to simulate data as well as implementations of feature extractor and classifier techniques. This architecture simulates requests being passed from a user into the multiagent system for a set of streaming data to be collected, refined, and classified. Once an event has been classified, an appropriate alert (if any) is sent back to the user.

3.1.1 Contract Net Protocol.

In order to facilitate this process, a control system was created to handle the flow of data into and out of the system. This multiagent system uses the previously discussed Contract Net Protocol as a foundation for creating the controlled environment while also setting up a pipeline for the flow of data. Figure 11 shows the

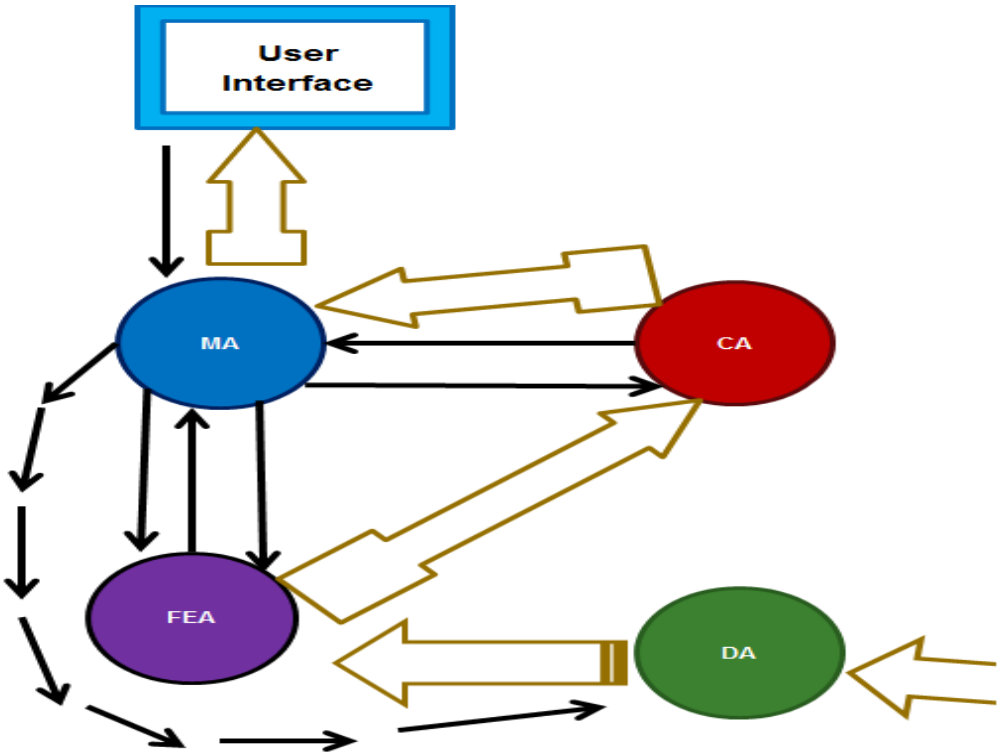


Figure 11. Date Flow.

flow of messages, including those used by the Contract Net Protocol, initiated by and terminating with the user. A request comes in from the user for a classification to be sent based on the streaming data from the current time domain, using an appropriate classification technique.

The Monitor Agent receives this request and sends out a CallForProposal to all available Classifier Agents to find one that is able to handle classification in the current context. Each available Classifier Agent responds with a bid based on the technique for which it advocates. The Monitor Agent chooses the first available Classifier Agent that advocates for an appropriate technique. Once a Classifier Agent has been selected, that agent responds to the Monitor Agent with the features that it will need in order to utilize the technique for which it advocates. The Monitor Agent then sends out another CallForProposal, this time to available Feature Extraction Agents, to find an agent that advocates for the techniques needed to pull the required feature values in appropriate way from the raw data for classification. Once selected, the Feature Extraction Agent is linked with the Data Agent that is receiving the streaming data from the wireless sensor nodes. The Feature Extraction Agent then utilizes feature extraction applications to pull the desired features from the raw data. These features are then communicated to the Classifier Agent so that the selected technique can be performed. The Classifier Agent then sends the classification to the initiating Monitor Agent, which then sends the classification to the requesting user.

The classification and feature extraction techniques are Python applications that the respective agents control. This allows the agents to act as advocates for their respective techniques instead of implementing the techniques directly. Data is passed from application to application via data sockets and the results are passed up to the initiating agent or user. This structure has been set up so that agents are able to be added and deleted from the system while

still maintaining the flow of data and accurately classifying events on a proactive or reactive basis. Allowing data and information to be communicated using sockets rather than ACL messages lessens the burden on the agents. This creates an environment where agents are able to achieve a higher level of performance.

Figure 12 is a snapshot of one run of this sequence as provided by Sniffer Agent. The Sniffer Agent is one of the agents packaged with the JADE. This agent allows the user to see the exchange of messages among agents. Lines 28 through 41 of the snapshot show the messages

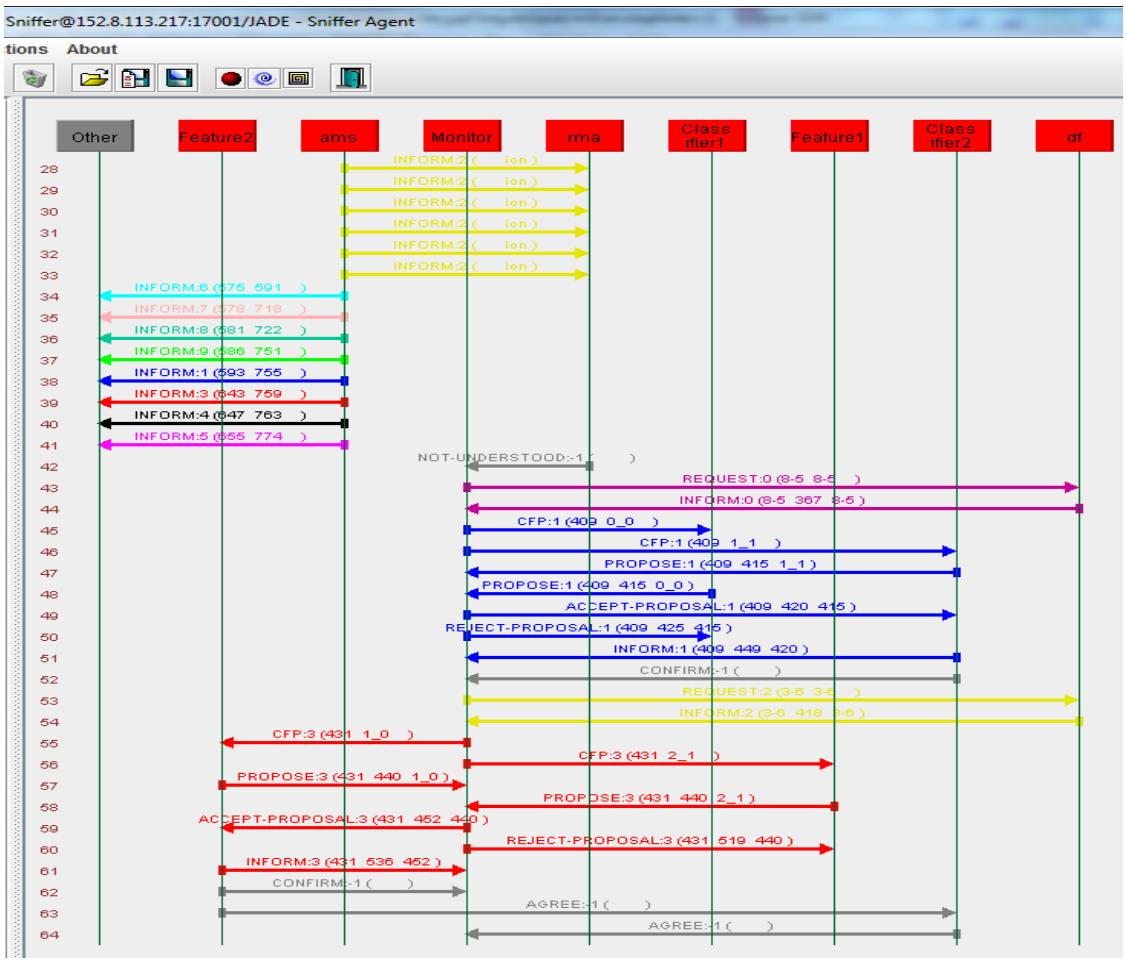


Figure 12. Sniffer Agent GUI of Contract Net Protocol.

passed between agents to set up the initial platform. In Figure 12, we see a message from the RMA (Remote Monitoring Agent) received by the Monitor Agent. The RMA allows for ACL

messages to be passed to agents from either agents outside of the local container or directly from the user. Once the Monitor Agent receives the message, it contacts the DF to find available Classifier Agents. After the Monitor Agent receives the message from the RMA, it sends a CallForProposal to both Classifier 1 and Classifier 2. The Classifier Agents both respond with proposals that include their bid. The Monitor Agent rejects the proposal of Classifier1 and accepts the proposal of Classifier 2. Next, Classifier 2 sends an INFORM followed by a CONFIRM message that includes its feature requirements. The Monitor Agent contacts the DF again to find available Feature Extraction Agents, and it uses the requirements provided by Classifier 2 to determine the best fit. Feature1 and Feature2 are sent a CallForPorposal message. They each respond with Propose messages that include their bid. Feature2's bid is accepted, and Feature1's bid is rejected. Then Feature 2 sends a Confirm and an Inform message to the Monitor Agent to verify its acceptance and then sends a message to Classifier2 to confirm the pipeline through which the data stream will flow. Finally, Classifier 2 sends a message to the Monitor Agent to initialize the data streaming process from the Data Agent.

3.1.1 Feature extraction and event classification (machine learning). In this research, our intent was to explore various data-driven techniques and to compare various machine learning techniques for the suitability for structural health monitoring using acoustic emissions. We tested these techniques on different data sets with a variety of characteristics. We were interested in assessing the accuracy of the diagnosis and prognosis of different machine learning techniques.

Initially, there were to be three machine learning ore related techniques researched, k-means, principle component analysis, and self-organizing maps. Machine learning techniques are implemented using NumPy, (and SciPy modules as needed) due to the availability of

mathematic functions in the NumPy library. Each of machine learning technique is used to create classification applications that are advocated for by the Classifier Agents.

To train the classifiers, they are fed relevant data sets, acquired from mechanical engineering collaborators through experiments where data outputs are stored in files, from which one may derive accurately calibrated classifiers. Each learning application will start out with a set of training data. Once acceptable results have been reached, the application will be used to attempt to classify a set of test data. After the test data shows positive results, the classifier used to produce these results is stored in an application for use in classifying live events.

3.1.2 Source to base station data flow. The next part of the implemented architecture is the flow of data between source and base station via Imote2 sensor nodes. In the final system,

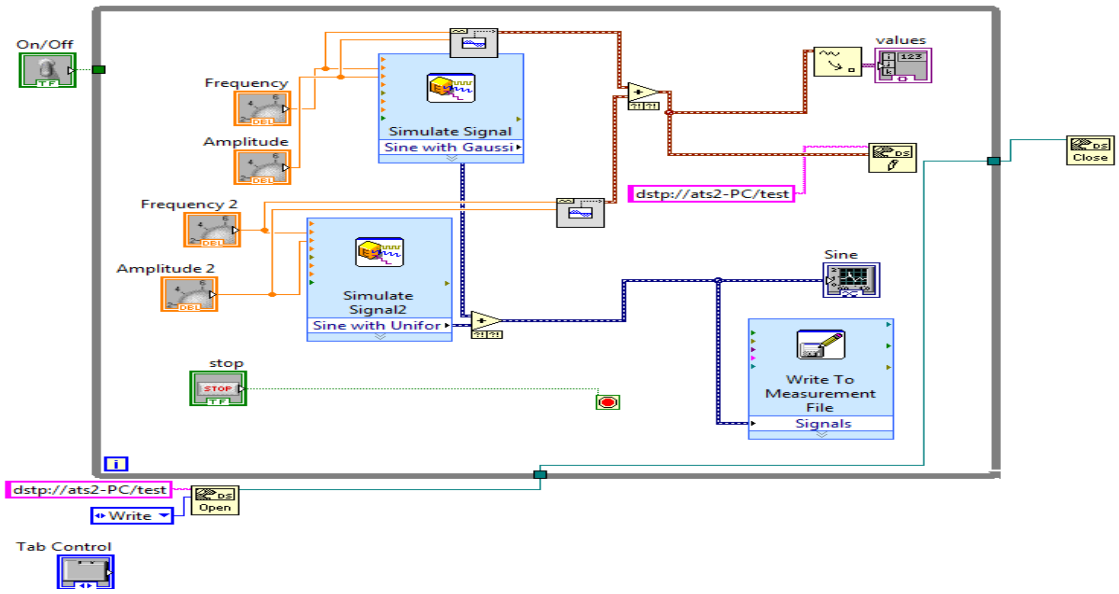


Figure 13. LabView Simulated Acoustic Emission VI (Configuration File).

sensor nodes will be placed in strategic locations of the aircraft that are as areas likely to experience structural failure or where failure would threaten the integrity of the entire aircraft. In Phase 1, two Imote2 devices have been programmed using nesC to receive data through a socket that is streamed from a simulation application on a data source station and sent to the base station

where the multiagent system is located.

The simulated data stream has been created using the LabView application. Figures 13 and 14 are screenshots of the implemented LabView application. The LabView application combines two sinusoid waves into one and communicates the characteristics over a data socket. The application also stores the characteristics of the created sinusoid wave in a file for review.

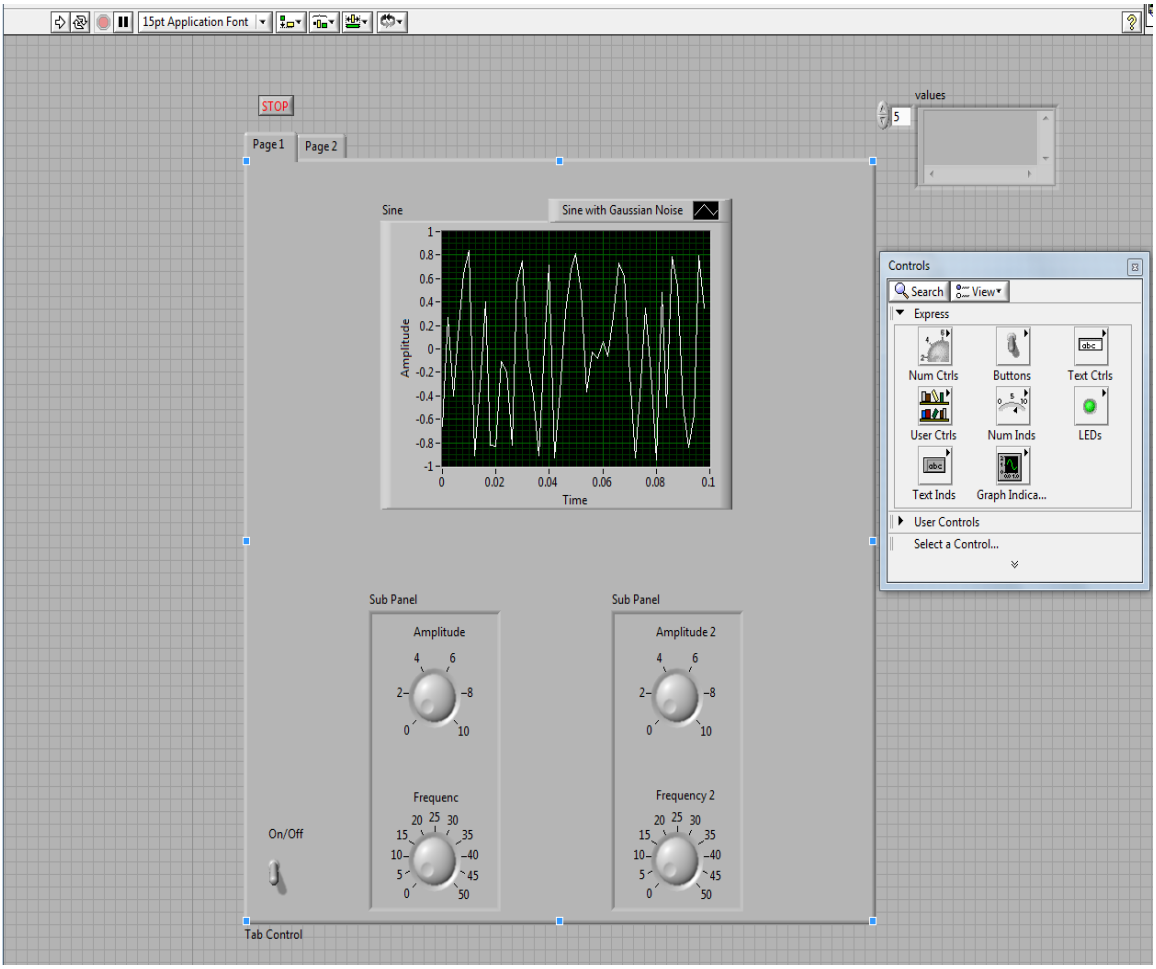


Figure 14. Simulated Acoustic Emission Sine Wave.

An intermediary Java program has been used to read data from the file and send it to the Imote2 nodes. The application is currently being used to test the lines of communication from source to base station and the flow of data. In the next version, the characteristics of the wave will more closely mirror those that will be present in during deployment.

Phase 1 provides a solid architecture that can be built upon to reach the goal of a fully realized structural health monitoring system. Each element of this current prototype will be present in the final system in some form. These linked systems each represent a key element in the final system and will allow for a critical test bed to test the development of the system incrementally.

CHAPTER 4

Discussion

In this section, the results of the implemented system architecture are discussed. This section also reviews the benefits and key aspects of this system that make it suited for use in structural health monitoring of an aircraft. First, this section discusses the benefits and challenges of using a multiagent system to control the flow of data, requests and alerts throughout the system. Second, the benefits and challenges associated implementing a multiagent system using the JADE development framework are reviewed. Finally, the benefits and challenges associated with using the Imote2 sensor nodes as a platform for communicating data streams to the base station.

4.1 Agent Programming

Monitoring the structural integrity of an aircraft while in motion requires a system that is able to adapt to varying internal and external stimuli at a moment's notice. This is a life critical necessity that could be the difference between a possible structural health concern being addressed and remediated in time or an inflight structural failure. Agent programming, in particular multiagent systems as previously discussed, are ideal control mechanisms that are able to meet the needs of such a system.

Agent programming allows for a system architecture that is inherently adaptable, distributed and robust. As a subset of artificial intelligence, agents are ideal for achieving complex tasks. With a mixture of object-oriented, multithreaded programming and event-driven programming, agent-oriented programming, as discussed in the background, are able to break down large tasks into smaller, more manageable tasks through the use of cooperative techniques such as the implemented Contract Net Protocol.

Utilizing the Monitor Agent, Feature Extraction Agents, Classifier Agents and Data Agent set up, each agent type is able to perform a specific task critical to the flow of data and event classification. This allows for distribution or tasks. Each agent of a specific type is able to perform the same task as other agents in the same group. This allows for the possibility of an agent that dies or become otherwise unresponsive being replaced by another available agent. Also, because agents are not strictly interdependent of one another, additional agents are able to be added into the architecture and perform their tasks, cooperating with agents already active in the system. Being the most utilized and accessible agent, as well as the agent in direct contact with the user as well as the Data Agent, the Monitor Agent needs to be able to process its tasks dynamically while sticking to a predetermined controlled order of operations. This was achieved through the use of certain built in mechanisms in the JADE platform.

4.2 JADE

Though agents are uniquely suited to perform the required actions to control a structural health monitoring system, effectively creating a multiagent system has its challenges. As previously stated, agent programming is a mixture of object-oriented, multithreaded programming and event-driven programming. Many resource-based and timing issues may easily present themselves if the design of the application is not approached in the correct manner. A suitable programming environment, with a tested library of classes, had to be chosen to allow for ease of development. JADE provided an ideal environment for which to create the implemented multiagent system.

Using JADE to develop the multiagent system allowed for a much smaller initial startup time due to it being based in Java. This also meant that any developed application could be distributable among the vast majority of computer hardware. Though the JADE environment was

relatively easy to begin coding in, effectively setting up agents that would interoperate while existing independently of each other posed unique challenges within the bounds of the available framework.

The main issues that arose in implementing the control system using JADE agents came from creating and allocating agent behaviors. Due to the nature of the system and the shared resources, certain agent behaviors needed to be processed in a precise order. For this, the implementation drew from the work done by Yolanda Jones in the area of Gaia schemas [42]. Gaia [21] is an agent-oriented design methodology for structuring individual agents as well as agents interacting in a multiagent system. The agent role schemas were created for each agent to design the individual roles, actions and interactions within the multiagent system. Figure 13 below shows one of the created role schemas.

Data Role Schema

Role Schema: DATA
Description: The main function of this agent is to review and store streaming data sources to be utilized by other agents while also directly controlling the functionality and keeping track of the status of the individual sensors that make up a given mesh network.
Protocols and Activities: SEND_DATA$_{FE}$,PULL_DATA, PUSH_DATA, PULL_STATUS, PUSH_STATUS
Permissions: reads *Mote Status // reads the status of the physical mote* *Data // reads the original data* changes *Raw Data Store // can store the raw data* *Mote Settings // can store, retrieve, and change the mote settings* *Data Storage Policy // can update data storage policy*
Responsibilities Liveness: DATA = (SEND_DATA$_{FE}$ | PULL_DATA | PUSH_DATA | PULL_STATUS | PUSH_STATUS)$^{\omega}$ PULL_DATA = receiveRequestData.wakeUpMotes.sendDataNotice.storeData PUSH_DATA = detectMotesActive.sendDataNotice.storeData SEND_DATA$_{FE}$ = receiveRequestDataAcess.sendGrantAcess.receiveDataAccessEnd PULL_SENSOR_STATUS = receiveRequestSensorStatus.collectStatus.sendSensorStatus PUSH_SENSOR_STATUS = detectExceptionCondition.sendSensorStatus Safety: • If ((mote_status ==awake) && (data > 0)) then store(data, dataStore, storage_policy) • If set_mote(MS) then $\bigcirc$ (mote_settings = MS) ((mote_status.awake)^{T1}.(mote_status.awake)^{T2}))$^{\omega}$, T2 >> T1

Figure 15. Data Agent Gaia Role Schema [42].

This schema was created to model the Data Agent. Schemas are broken down into five sections. The first section states what agent role this schema is for. The next section, Description, provides a summary of the role's tasks and overall functionality. The Protocols and Activities section provides a high level list of the actions the agent will be able to take. The Permissions section dictates what data the agent can read versus what data it can change. The Responsibilities section, broken down into Liveness and Safety, first lists the steps the agent will take to process the Protocols and Activities and, second, lists the conditions the agent must not violate when processing these actions. The Gaia two-step process of analysis and design allowed us to create a well thought out model and point of reference for creating the agents with their eventual dynamic interaction in mind.

Coding certain behaviors needed to be done in a tiered manner, where specified behaviors run in a strict order with encoded sub-behaviors also running in a predetermined order. This was facilitated through the use of the tiered SequentialBehaviour class, one of the behavior types available in the JADE framework. SequentialBehaviour, as the name implies, sets a strict order for included sub-behaviors. Sub-behaviors run in the order in which they are added. For the purpose of this system, stricter control was achieved through the use of sequential sub-behaviors.

Because of the vast amounts of data processed in real time, it was essential to create a subcontracting structure that would allow for tasks to be efficiently processed by available agents. For this purpose, the Contract Net Protocol (CNP) messaging template provided in JADE was ideal. As previously discussed, the CNP is the primary mechanism used in breaking down large tasks and subcontracting those individual responsibilities to available agents. In JADE, the developer is able to create messaging templates that dictate certain messages being communicated as part of the propose-bid-award-accept process of the CNP. Utilizing this allows

the Monitor Agents to dynamically determine the best agent for pulling needed data and classifying real time events, making this ideal for use in the structural health monitoring system

4.3 Wireless Sensor Network

As previously discussed, a wireless sensor network is comprised of two or more wireless sensor nodes that communicate within a network. For the purposes of this implementation, the minimum number of nodes in a network was used. The primary reason for this was to validate lines of communication as well as the functionality of the nodes. This also provided a base from which to build off of.

The major benefits of using the Imote2 is the mote's stack ability and interface. This ability allows us to link several different types of sensors to a single node to more closely simulate the sensing abilities and requirements that will be present in the fully realized deployed system. The current plan of action is to link the motes to piezoelectric sensors to sense vibrations as well as a newly developed sensor created by the Department of Mechanical Engineering at North Carolina A&T State University. Though the Imote2 will not be able to handle the full stream of data, it will be able to communicate enough of it to create a clear picture of the situation and state of the implementation, providing direction for the next step in development, including future hardware for improved processing power and software that would allow processing to be done at the mote level before it is sent to the multiagent system.

A challenge in creating this sensor network was a lack of programming boards. The programming board is used to link an Imote2 directly to a computer. The initial plan was to have one sensor node linked to a computer that produces the simulated raw data while another is linked to the base station computer providing the multiagent system with access to the data stream. The Imote2 devices would transfer data wirelessly from the data source to the base

station to create a basic wireless network architecture that allows the system to test the base case for data origination and communication. Though we had four Imote2 devices, we were only fortunate enough to have one programming board. Through the assistance of PhD candidate William Wright, a workaround was discovered. We were able to utilize xBee radios, which also transfer data wirelessly using a version of the 802.15.4 wireless communication protocol to facilitate transfer of data from the data source to the base station. This allowed us to verify the wireless communication ability of the Imote2, and it provides a workaround process in cases where certain equipment is not present in the system.

CHAPTER 5

Conclusion

In conclusion, the goal of this research was to test the viability of utilizing a multiagent system in implementing a sensor web-based structural health monitoring system. The technologies reviewed here were researched and tested with the aim of creating such a sensor web. The architecture discussed, which utilizes a multiagent system for control of a structural health monitoring sensor web, will utilize pre-calibrated classifiers created from machine learning techniques to characterize acoustic emissions into identifiable categories. This collaborative sensor web will allow for a flexible and robust system that will be able to adapt to both standard and unforeseen events by breaking down tasks, allowing for large tasks to be more efficiently handled. Though this proposed approach of an agent-based structural health monitoring system was primarily for the benefit of the NASA-CAS (National Aeronautics and Space Administration Center for Aviation Safety) at A&T, there are numerous industries and applications that could greatly benefit from a similarly architecture.

The aim of this architecture is to create an autonomous system that takes some of the burden off personnel monitoring the structural integrity of an aircraft. By creating inferences as to the structural health of a system based on the acoustic emissions received from piezoelectric sensors, in essence, the wireless sensor network will act as skin. This will allow the characteristics and locations of acoustic emission sources to be more accurately relayed to controllers in a timely manner because relevant data will be continuously communicated, processed and characterized via agents. In the end, adding this system to the safety processes already in place for monitoring the structural health of airplanes will allow for a much more reliable and safe system.

5.1 Future Work

There are a few distinct areas that can be improved upon in a future implementation to further test the viability of utilizing an agent control structure for communicating requests, alerts, and data as well as making changes to the data transfer setup of the wireless sensor network. Figure 16 illustrates some of the changes that will be included in a future implementation of this system. Comparing the implementation in Figure 16 to that of Figure 10, quite a few differences can be observed.

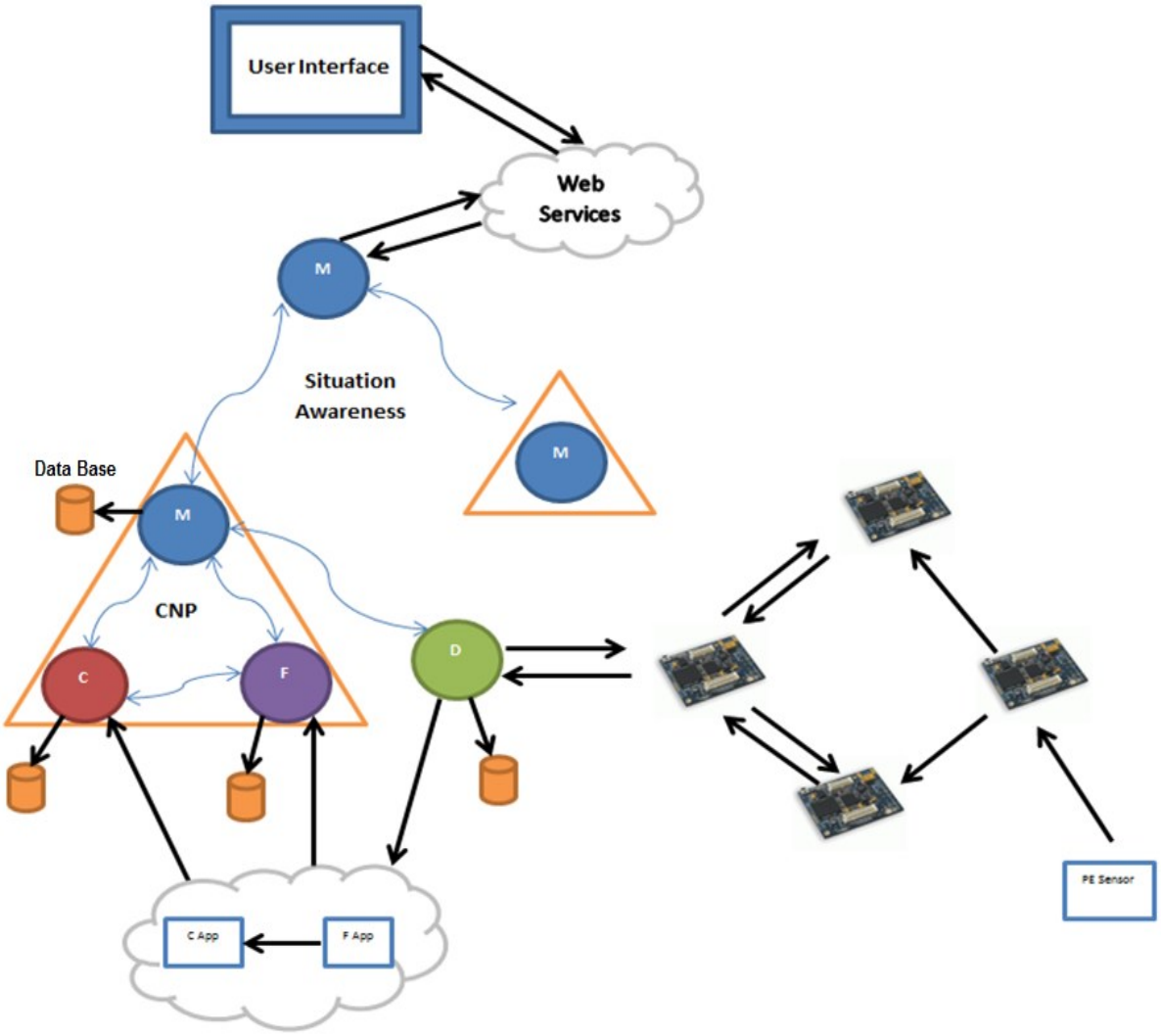


Figure 16. Future System Architecture.

The primary focus of this future implementation will be on creating a tiered hierarchy of monitor agents that are able to draw inferences about the overall health of the system based on analysis of localized areas. The higher up the hierarchy, the more global the agent's view will be of the overall system health, while the lower level agents will have access directly to data.

To dynamically configure and reconfigure the hierarchy of Monitor Agents based on the services they provide and the current needs of the system, the CNP will be used. This will provide the monitoring system with a greater level of flexibility and adaptability as the system will be able to determine the optimum configuration to handle the needs of the system and proactively initiate changes in order to effectively and accurately work under varying conditions in changing environments.

Other future work will involve the inclusion of web services. Including web services addresses accessibility and system resource limitations. The first place where web services will be added is in the form of classifier and feature extractor applications. These applications will be created using a python web service framework, such as CherryPy, to facilitate open and direct access to the services from agents and users. Data will be communicated to these applications using sockets which will be controlled by the Monitor and Data Agents. Allowing agents to access applications through the web will make it easier to update the system with new classification and feature extraction techniques.

Controlling the flow of data through and between applications is a key concern. The flow of data through the sockets between applications will be controlled by the Agents. Once the pipeline between a Feature Extraction Agent, a Classifier Agent, and the Monitor Agent is determined, the Data Agent will contact the Monitor Agent with the port through which the raw data stream will be passed. This port ID is communicated to the Feature Extraction Agent which

connects its advocated application to the socket using the designated port. Once the feature extraction application has the data stream, a new socket is opened to pass the feature vectors to the classifier application. The Feature Extraction Agent will contact the Classifier Agent and communicate the port through which this socket is streaming the vectors. The Classifier Agent then connects its advocated application to the port via this port ID so that the classifier application is able to classifier the event. This classification is then passed back to the Classifier Agent, which will in turn pass the classification to the Monitor Agent via an ACL message. This classification will be passed up the hierarchy to the initiating monitor or user and also stored in a database for later review. We are currently experimenting with the use of instances of a light-weight Web server (CherryPy) as a way for agents to interact with the Python applications to set up sockets as required for the communication discussed here.

Web services will be used to provide users with direct access to specified agents. This will be facilitated using the Web Service Integration Gateway (WSIG). WSIG is a JADE add-on that allows agents to be published as web services. This would give the user the ability to directly access individual registered agents, making checking the status of a particular section of the structure from a remote location much easier.

Another enhancement will be increasing the number of sensor nodes and switching their data stream from simulated LabView data to streaming data from piezoelectric and custom sensors. Adding additional sensor nodes into the wireless network (as indicated in Figure 16) allows us to test out the network's ability to support multi-hop functionality. Data will travel from a source location via a connected Imote2 through a chain of one or more available intermediary nodes. These nodes will transfer the data stream along with their distinct ID eventually to an Imote2 node connected to the base station. This will allow us to test the ability

of data to be passed through multiple chains of nodes from source to destination. Also, passing data to the base station with unique mote IDs will allow us to simulate data being streamed from different areas of the structure. This will allow us to test sending commands to the motes, such as sleep or wakeup. Additionally, this will allow us to address concerns regarding nodes shutting down and data streams needing to be rerouted.

Imote2 sensor nodes where chosen both for their wireless data transfer abilities as well as their stackable properties. Switching the Imote2's from simply passing simulated data to pulling real-time actual data through the collaboration with students in the Mechanical Engineering Department will allow the system to be tested under conditions that are much closer to the conditions and range of data points that will be seen in the deployment environment. Though the bit transfer rate of the Imote2Sensor nodes may not be sufficient enough to handle all of the data that will be streaming through the system in the final version, at least this will allow the system, in particular the multiagent system, and the validity of the classifiers to be tested under conditions much closer that of the final version.

Data that warrants review will be stored in databases at each level of data processing. Finally, a user dashboard will be linked to the system to increase the level of user access and control while also presenting alerts and classifications in a more user friendly manner. These future enhancements will all serve as milestones in an attempt to implement the final goal of a fully realized structural health monitoring system controlled by a network of interactive, reactive and proactive software agents providing real time alerts and vital data to user in the aircraft while also allowing for remote access to offsite technicians.

References

[01] Sun SPOT World, Sunspot world program the world, (2010, January). Retrieved March 20, 2013, from http://sunspotworld.com/vision.html

[02] Caicedo,A., The Sun Small Programmable Object Technology(Sun SPOT): Java™ Technology-Based Wireless Sensor Networks, http://www.austinjug.org/presentations/SunSpots.pdf

[03] MICA z Wireless Measurement System, Crossbow, Retrieved March 20, 2013, from http://www.docstoc.com/docs/20049970/MICAz-Datasheet

[04] Kephart, J.O. & Chess, D. M. (2003). The Vision of Autonomic Computing. *Computer*, vol. 36, no. 1, 2003, pp. 41–50.

[05] Luck, M & McBurney, P & Preist, C (2003, January). Agent Technology: Enabling Next Generation Computing, AgentLinkII, http://www.agentlink.org/roadmap/al2/roadmap.pdf

[06] Labrou, Y. & Finin, T., Semantics and Conversations for an Agent Communication Language, http://arxiv.org/pdf/cs.MA/9809034.pdf

[07] Boella, G. & Damiano, R. & Hulstijn, J. & VanderTorre (2006). L., A Common Ontology of Agent Communication Languages: Modeling Mental Attitudes and Social Commitments using Roles, http://www.di.unito.it/~guido/articoli/30-ao07b.pdf

[08] Vasudevan, V. (1998). Comparing Agent Communication Languages, http://act-r.psy.cmu.edu/~douglass/Douglass/Agents/TopicPapers/KQML/9807-comparing-ACLs.pdf

[09] Wooldridge, M. (2009, June 22). An Introduction to MultiAgent Systems, 2nd edition Wiley

[10] Davis, R., and Smith, R.G. (1988). Negotiation as a Metaphor for Distributed Problem Solving. In: Bond, A., and Gasser, L. eds. Readings in Distributed Artificial Intelligence , 333-356. San Mateo, Calif.: Morgan Kaufmann.

[11] The Foundation for Intelligent Physical Agents (2012). Retrieved March 20, 2013, from http://www.fipa.org/

[12] The Cricket Indoor Location System, An NMS Project, (2006, March 13), Retrieved March 20, 2013, from http://cricket.csail.mit.edu/

[13] Imote2, HIGH-PERFORMANCE WIRELESS SENSOR NETWORK NODE, http://bullseye.xbow.com:81/Products/Product_pdf_files/Wireless_pdf/Imote2_Datasheet.pdf

[14] Rice, J. A. & Spencer, B.F. Jr. (2008). Structural health monitoring sensor development for the Imote2 platform http://sstl.cee.illinois.edu/papers/SPIE08_Rice_Paper.pdf

[15] Liggins, M. E. & Hall, D. L. & Llinas, J.(2008). *Multisensor Data Fusion*, Second Edition. Theory and Practice (Multisensor Data Fusion). CRC,

[16] Hall, D. L. & McMullen, S. A. H. (2004). Mathematical Techniques in Multisensor Data Fusion

[17] National Aeronautics and Space Administration, Data Driven Prognostics, Retrieved March 20, 2013, from http://ti.arc.nasa.gov/tech/dash/pcoe/data-driven-prognostics/

[18] Fok, C. L. (2008, April 9). Agilla, A Mobile Agent Middleware for Wireless Sensor Networks, Retrieved March 20, 2013, from http://mobilab.cse.wustl.edu/projects/agilla/

[19] Lange, D. B. & Oshima, M(1999, March).Seven Good Reasons for Mobile Agents, Communications of ACM, , Vol. 42, No. 3

[20] Mobile Agents, Intelligent Assistants on the Internet, http://www.itswtech.org/Lec/Manal%28system%20programming%29/simeners_B/Mobile_Agent.pdf

[21] Wooldridge, M. & Jennings, N. & Kenney, D.(2000). The Gaia Methodology for Agent-Oriented Analysis and Design, Kluwer Academic Publishers, Boston,

[22] FIPA, The Foundation for Intelligent Physical Agents, IEEE Foundation for Intelligent Physical Agents, (2012). Retrieved March 20, 2013, from http://www.fipa.org/

[23] Friedman-Hill, E.(2003). JESS in action, Rule-Based system in Java

[24] Reinhardt, A.(2009). Exploiting Platform Heterogeneity in Wireless Sensor Networks for Cooperative Data Processing, Retrieved March 20, 2013, from http://www.ti5.tu-harburg.de/events/fgsn09/proceedings/fgsn_021_slides.pdf

[25] Wang, P.(2010). Python Evangelism 101 or Python Meets The Enterprise, conference.scipy.org/scipy2010/slides/lightning/peter_wang_python_evangelism.pdf

[26] NumPy, (2012). Retrieved March 20, 2013 http://numpy.scipy.org/

[27] OnLamp.com Python DevCenter, Broadcasting with Python, (2000, September 27), Retrieved March 20, 2013, from http://onlamp.com/pub/a/python/2000/09/27/numerically.html

[28] deBuyl , P(2012, March 23)., Retrieved March 20, 2013, from http://www.scipy.org/

[29] Hall, D. & Llinas, J., An Introduction to Multisensor Data Fusion, http://www.hh.se/download/18.70cf2e49129168da0158000134101/hall-multisensor-datafusion.pdf

[30] TinyOS, (2010). Retrieved March 20, 2013, from http://www.tinyos.net/

[31] Levis, P. (2006, January). TinyOS Programming, Retrieved March 20, 2013, from http://csl.stanford.edu/~pal/pubs/tinyos-programming.pdf

[32] Karlof, C. & Sastry, N. & Wagner, D., TinySec: A Link Layer Security Architecture for Wireless Sensor Networks, Retrieved March 20, 2013, from http://www.cs.berkeley.edu/~daw/papers/tinysec-sensys04.pdf

[33] Delin, K. A. (2004, April 1). The Sensor Web: A Distributed, Wireless Monitoring System, Retrieved March 20, 2013, from http://www.sensorsmag.com/networking-communications/the-sensor-web-a distributed-wireless-monitoring-system-942

[34] Reinhardt, A.(2009)., Exploiting Platform Heterogeneity in Wireless Sensor Networks for Cooperative Data Processing, Retrieved March 20, 2013, from http://www.ti5.tu-harburg.de/events/fgsn09/proceedings/fgsn_021_slides.pdf

[35] IEEE Advancing Technology for Humanity, 2012, Retrieved March 20, 2013, from http://www.ieee.org

[36] Ying-Darlin (1998). On IEEE 802.14 Medium Access Control Protocol, Retrieved March 20, 2013, from http://speed.cis.nctu.edu.tw/~ydlin/80214.pdf

[37] Iyengar, S.(2010). *Fundamentals of Sensor Network Programming*. Wiley. pp. 136.

[38] Lee, J. & Su, Y. & Shen, C. (2007, November 8). A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee and Wi-Fi, The 33rd Annual Conference of the IEEE Industrial Electronics Society (IECON)

[39] Gutierrez, J. A. (2005). IEEE Std. 802.15.4, Enabling Pervasive Wireless Sensor Networks, Retrieved March 20, 2013, from http://www.cs.berkeley.edu/~prabal/teaching/cs294-11-f05/slides/day21.pdf

[40] Marsland, S.(2009). Machine Learning, An Algorithmic Perspective, 2009

[41] Cambridge LabView User Group, (2012), Retrieved March 20, 2013, from http://www.labviewcambridge.co.uk/

[42] Jones, Y. B.(2011). Sensor Webs for Structural Health Monitoring of Aircraft, 2011

[43] Zambonelli, Franco, Jennings, Nicholas R., & Wooldridge, Michael. (2003, July). Developing Multiagent Systems: The Gaia Methodology. *ACM Transactions on Software Engineering and Methodology*, Vol. 12, No. 3, pp. 317–370, Retrieved March 20, 2013, from http://users.ecs.soton.ac.uk/nrj/download-files/tosem03.pdf

[44] Jade, Java Agent Development Framework , an Open Source platform for peer-to-peer agent based applications, http://jade.tilab.com/

[45] Javadi, F. & Munasinghe, K. S.(2010, August 1). *Wireless Communications and Mobile Computing,* A fast and reliable routing technique for wireless mesh networks, 12(9) 782-796, doi 10.1002/wcm.1013

[46] ICTs: device to device communication. Retrieved March 20, 2013, from Open Learn LabSpace, http://labspace.open.ac.uk/mod/resource/view.php?id=369305

[47] Structural Health Monitoring Workshop, Sponsored by Army, Navy, and Air Force, Administered by the Advanced Materials, Manufacturing & Testing Information Analysis Center (AMMTIAC), November 18-20, 2008, Austin, TX

[48] Acoustic Emission Research (2012). Retrieved March 20, 2013, from Physical Acoustics Corporation, http://www.pacndt.com/index.aspx?go=research&focus=/capabilities/ae%20research.htm

Appendix A

Monitor Agent

/*Monitor Agent Code- primary agent for facilitating cooperation between agents in the system.

In direct contact with the user and the Data Agent*/

```
package main;
import java.io.IOException;
import java.util.Random;
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.Behaviour;
import jade.core.behaviours.CyclicBehaviour;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.SequentialBehaviour;
import jade.core.behaviours.TickerBehaviour;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.wrapper.AgentController;
import jade.wrapper.PlatformController;
import javax.swing.JTree;
import javax.swing.JOptionPane;
import javax.swing.tree.DefaultMutableTreeNode;

public class Monitor extends Agent {

//    static JTree MAS;
      String name;
//    DefaultMutableTreeNode node;
      String cOffer;
      String feOffer;
      AID featureExt;
      AID classifier;
      AID user;
      String features;
      boolean done;
      boolean classified;
      public SequentialBehaviour seq1;
      public SequentialBehaviour seq2;
      public SequentialBehaviour seq ;
      public CyclicBehaviour cyc;

      protected void setup(){
            setCOffer();
            done = false;
            classified = false;
            try{
               if(getArguments()!=null){Object[] args = getArguments();
                 cOffer = args[0].toString();}
            }
            catch(ArrayIndexOutOfBoundsException e){
```

```
		}
		DFAgentDescription dfd = new DFAgentDescription();
		dfd.setName(this.getAID());
		addBehaviour(new DFRegistration(this, "Monitor", "base", dfd));

		//node = new DefaultMutableTreeNode(this);
		//MAS = new JTree(node);
		//System.out.println("Monitor "+getLocalName()+ " Adding
			Behaviours...");
		TickerBehaviour tick = new TickerBehaviour(this, 1000){
			long startTime = System.currentTimeMillis();

			@Override
			protected void onTick() {

				if ((System.currentTimeMillis()-startTime) >=
	1000)stop();

			}
		};

		seq1= new SequentialBehaviour();


//
		//if (receive != null){
	//	System.out.println(receive.getContent());
		//setCOffer(receive.getContent());

		seq1.addSubBehaviour(new CNP(this, "Classification"));
		seq1.addSubBehaviour (
				new TickerBehaviour(this,1000){

					@Override
					protected void onTick() {
						if (done == true){
							seq1.addSubBehaviour(new
								CNPResultBehaviour());
							done = false;
							stop();
						 }

					}

				});



		seq2= new SequentialBehaviour();
		seq2.addSubBehaviour(new CNP(this, "Feature Extraction"));
		seq2.addSubBehaviour (
			new TickerBehaviour(this,1000){

				@Override
				protected void onTick() {
					if (done == true){
```

```
                              seq2.addSubBehaviour(new
                                    CNPResultBehaviour());
                              stop();
                        }
                  }

            });

      //seq2.addSubBehaviour(tick);
      /*
      OneShotBehaviour finishBehaviour = new OneShotBehaviour(){
            public void action(){
                  //System.out.println("\n9999  ^&(&*^*&  9999\n");
            //    while(true){
                        if (done == false) this.block();
                                          ACLMessage receiveMsg =
                  myAgent.receive(MessageTemplate.MatchPerformative(ACL
                  Message.AGREE));//blockingReceive(MessageTemplate.Mat
                  chPerformative(ACLMessage.AGREE));
                        //if (receiveMsg!= null)
                  System.out.println("\n"+"Event Classification
                        Results: " + receiveMsg.getContent());
                  classified = true;
                  //    break;
                  //    else this.block();

                  //}
                  super.block();
            }
      };
      */
      seq = new SequentialBehaviour();
      seq.addSubBehaviour(new OneShotBehaviour(){
            public void action(){
                  classified = false;
                  System.out.println("Made it into seq ");
                  ACLMessage receive =
                        blockingReceive(MessageTemplate.MatchPerformati
                        ve(ACLMessage.NOT_UNDERSTOOD));
                  user = receive.getSender();
                  setCOffer(receive.getContent());
                  //block();
            }
      });
      seq.addSubBehaviour(seq1);
      //seq.addSubBehaviour(new OneShotBehaviour(){public void action
(){System.out.println("\nSeq: Started one second wait\n");}});
      seq.addSubBehaviour(tick);
      //seq.addSubBehaviour(new OneShotBehaviour(){public void action
(){System.out.println("\nSeq: Finished one second wait\n");}});
      seq.addSubBehaviour(seq2);
      seq.addSubBehaviour(new OneShotBehaviour(){public void action
(){root().reset();}});
      //seq.addSubBehaviour(tick);
      //seq.addSubBehaviour(finishBehaviour);
```

```
      Looper loop = new Looper (this, seq);

}//end of setup
class Looper extends OneShotBehaviour{//
      Behaviour bev;
      int count = 1;
    public Looper(Agent a, Behaviour b) {
      super(a);
      addBehaviour(b);
      bev = b;
    }

    public void action() {}

  }

public void finalMethod(){

      addBehaviour (new OneShotBehaviour(){
            public void action(){
                  //System.out.println("\n9999  ^&(&*^*&  9999\n");
                  //while(true){
                  ACLMessage receiveMsg =
                        myAgent.receive(MessageTemplate.MatchPerformati
                        ve(ACLMessage.AGREE));//blockingReceive(Message
                        Template.MatchPerformative(ACLMessage.AGREE));
                  //if (receiveMsg!= null)
                  System.out.println("\n"+"Event Classification
                        Results: " + receiveMsg.getContent());

                  //classified = true;
                  //break;
                  //removeBehaviour(seq);
            }
      });
}

protected String getCOffer() {//gets the Classifier Offer
      return cOffer;
}

protected void setCOffer(String offer) {//sets the Classifier Offer
      this.cOffer = offer;
}
protected void setCOffer() {//sets the Classifier Offer
      this.cOffer = "kMeans 5";
}
protected String getFEOffer() {//gets the Feature Extractor Offer
      return feOffer;
}

protected void setFEOffer(String offer) {//sets the Feature Extractor
Offer
      this.feOffer = offer;
}
```

```
	protected void setFeatureAID(AID FID) {//sets the Feature Extractor
	Offer
		this.featureExt = FID;
	}

	protected AID getFeatureAID() {//sets the Feature Extractor  Offer
		return featureExt;
	}

	protected void setClassifierAID(AID CID) {//sets the Feature Extractor
		//Offer
		this.classifier = CID;
	}

	protected AID getClassifierAID() {//sets the Feature Extractor  Offer
		return classifier;
	}
}//end of class file
```

Appendix B

CNP Behaviour

```
/*CNP Behaviour used by Monitor Agents to search the DF find agents that will
      be sent CallsForProposals */

package main;
import java.util.Date;
import java.util.Random;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.Agent;
import jade.domain.FIPAException;
import jade.domain.FIPANames;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.lang.acl.ACLMessage;


public class CNP extends OneShotBehaviour{
      Monitor agent;
      String service;
      ACLMessage cfp;

      public CNP(Monitor myAgent, String s){// constructor allocates agent as
      //the calling monitor and service as the passed through as s
            agent = myAgent;
            service = s;
      }


      public void action() {

            //Create Call for proposal (CFP) message for Data agents
            cfp = new ACLMessage(ACLMessage.CFP);
            cfp.setProtocol(FIPANames.InteractionProtocol.FIPA_CONTRACT_NET);
            cfp.setContent("BEGINNING CNP");

            System.out.println("Finished setting up CFP");
            System.out.println("CNP:  Beginning...");

            try {
                  //Find agents for Contract Net and send CFP message to them
                  //DFAgentDescription[] monitorsDescription =
                        DFUtils.searchAllByType(this, "Amount of Damage",
                        null);
                  DFAgentDescription dfd = new DFAgentDescription();
                  ServiceDescription sd = new ServiceDescription();
                  sd.setName(service);
                  dfd.addServices(sd);
                  DFAgentDescription[] monitorsDescription =
                        DFService.search(agent, dfd);

                  for(int i = 0; i< monitorsDescription.length; i++){
                        cfp.addReceiver(monitorsDescription[i].getName());
```

```
                        System.out.println(monitorsDescription[i].getName().g
                              etLocalName()+ "*...***");
                  }

                  System.out.println("Finishing CFP...");

                  agent.addBehaviour(new CNPBehaviour(agent, cfp));

            } catch (FIPAException e){
                  System.out.println("Services are "+service);
                  System.out.print(e);
            }
            reset();
      }
}
```

Appendix C

CNPBehaviour Behaviour

```
/*CNPBehaviour used by Monitor Agents extends the ContractNetInitiator and
      has built in functions associated with CNP to handle proposals,
      rejections, informs and failures*/

package main;

import java.io.IOException;
import java.util.Vector;
import jade.core.Agent;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.lang.acl.ACLMessage;
import jade.proto.ContractNetInitiator;

public class CNPBehaviour extends ContractNetInitiator {

      int counter = 0;
      String offer;

      public CNPBehaviour(Agent a, ACLMessage cfp) {
            super(a, cfp);
            System.out.println ("CNPBehaviour...");
      }
            // TODO Auto-generated constructor stub

      protected void handlePropose(ACLMessage msg, Vector acceptances) {
            System.out.println("Beginning CNPBehaviour");
            String bid = msg.getContent();

            System.out.println("Bid = " + bid);
            System.out.println("Agent considering a bid: " + bid);

            Monitor monitor = (Monitor)myAgent;
            ACLMessage response = msg.createReply();

            System.out.println(msg.getSender().getLocalName());

            if ((msg.getSender().getLocalName()).charAt(0) == 'C'){

                  System.out.println("Sender is a
                  Classifier..."+bid+"..."+((Monitor)myAgent ).getCOffer() );
                  if (bid.equalsIgnoreCase( ( (Monitor)myAgent ).getCOffer()
                        ) && counter == 0) {
                  System.out.println("Monitor and Classifier bids are
                        equal");
                  response.setPerformative(ACLMessage.ACCEPT_PROPOSAL);
                  response.setContent("You've got a deal!");

                  System.out.println(msg.getSender().getLocalName()+" got It
                        Out!!!\nCounter = "+ counter);
                  counter++;

            } else {
```

```
                System.out.println("Monitor and Classifier bids are
                      unequal");
                response.setPerformative(ACLMessage.REJECT_PROPOSAL);
                response.setContent("...think I'll pass on that one.");
              }
              myAgent.send(response);
              System.out.println("Out of CNPBehaviour: Classifier");
              monitor.done= true;

        }

        else{
              System.out.println("Sender is a Feature Extractor");
              if (bid.equalsIgnoreCase( ( (Monitor)myAgent
                    ).getFEOffer()) && counter == 0) {
              response.setPerformative(ACLMessage.ACCEPT_PROPOSAL);
              try {
                    response.setContentObject(monitor.getClassifierAID();

              } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
              }
              System.out.println(msg.getSender().getLocalName()+" got It
                    Out!!!\nCounter = "+ counter);
              counter++;
        }
        else {

              response.setPerformative(ACLMessage.REJECT_PROPOSAL);
              response.setContent("...I'll pass on that one.");
        }

        myAgent.send(response);
        System.out.println("Out of CNPBehaviour: Feature");

     }

  }

  protected void handleInform(ACLMessage msg) {
     System.out.println("Inform received: " + msg.getContent());
  }

  protected void handleRefuse(ACLMessage msg) {
     System.out.println("Refusal received: " + msg.getContent());
  }

  protected void handleFailure(ACLMessage msg) {
     System.out.println("Failure received: " + msg.getContent());
  }
}
```

Appendix D

CNPResultBehaviour Behaviour

```
/*CNPResultBehaviour is used by the Monitor Agent to process the results of
      the CNP and contact the winning Classifier or Feature Agent*/

package main;

import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.Behaviour;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.SimpleBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

public class CNPResultBehaviour extends OneShotBehaviour {
      Monitor agent;
      ACLMessage clMsg;
      static int counter = 1;
      public void action() {
            agent = (Monitor) myAgent;
            System.out.println("CNPResultsBehaviour:  Beginning...");
            //Create a message template for the ACL 'CONFIRM' performative
            MessageTemplate mt =
                  MessageTemplate.MatchPerformative(ACLMessage.CONFIRM);

            ACLMessage msg = myAgent.receive(mt);

            if (msg != null){
                  System.out.println("\n"+msg.getSender().getLocalName()+"\n"
                        );
                  System.out.println("msg not null");
                  if ((msg.getSender().getLocalName()).charAt(0) == 'C'){

                        System.out.println("%%Received message from
                              "+msg.getSender().getLocalName()+" \n"+
                              msg.getContent()+"\n");
                        agent.setFEOffer(msg.getContent());//sends the
                              requirements for the Classifier up to the
                              Monitor to be used in feature CNP
                        agent.setClassifierAID(msg.getSender());//saves the
                              Classifier's name for later use
                        agent.done=true;
                        return;
                  }// end of else if
                        ((msg.getSender().getLocalName()).charAt(0) == 'C')

                  else if ((msg.getSender().getLocalName()).charAt(0) ==
                        'F'){
                        System.out.println("%%Received message from
                              "+msg.getSender().getLocalName()+" \n"+
                              msg.getContent()+"\n");
      /*
                              ACLMessage respond = msg.createReply();
```

```
                              respond.setContent("Testing");
                              myAgent.send(respond);
      */
                        agent.finalMethod();
                        return;
                        } //end of else if
                              ((msg.getSender().getLocalName()).charAt(0) ==
                              'F'){
            }
            //else block();

      }//end of action block

}//end of CNPResultBehaviour block
```

Appendix E

CreateMonitor Behaviour

```
/*CreateMonitor Behavior used to allow an agent to Create a monitor*/

package main;

import javax.swing.JTree;
import jade.core.Agent;
import jade.core.behaviours.OneShotBehaviour;
import jade.wrapper.AgentController;
import jade.wrapper.PlatformController;
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;

public class CreateMonitor extends OneShotBehaviour{
      static int count;
      Agent monitor;
      protected DefaultMutableTreeNode node;

      public CreateMonitor(DefaultMutableTreeNode leaf, Agent a, int num){
            node = leaf;
            monitor = a;
      }

      public CreateMonitor(Agent a, DefaultMutableTreeNode leaf){
                  monitor = a;
                  node = leaf;
                  count = 0;
      }

      public void action(){
            count++;//increment the counter

            try{//create a new monitor agent in the main container
                  PlatformController container =
                        monitor.getContainerController();
                  AgentController monitor =
                        container.createNewAgent("Monitor"+count,
                        "main.Monitor", null);
                  monitor.start();
                  node.add(new DefaultMutableTreeNode(monitor));
            }catch(jade.wrapper.ControllerException err){
                  System.out.println("ERROR CREATING NEW AGENT: " + err);
            }//ends try/catch(jade.wrapper.ControllerException err){


      }//closes void action()

}
```

Appendix F

Data Agent

```
/*Data Agent code used to add behaviors for accessing raw data stream through
      socket and pass it to agent for review*/

package main;

import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.Behaviour;
import jade.core.behaviours.CyclicBehaviour;
import jade.core.behaviours.SequentialBehaviour;
import jade.core.behaviours.TickerBehaviour;
import jade.domain.FIPANames;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.lang.acl.*;
import jade.proto.ContractNetResponder;
import java.io.*;
import java.net.*;

public class Data extends Agent{
      ServerSocket socket=null;
      Socket ss_accept=null;
      static int []newNums;//stores individual feature vectors
      static int[]oldNums;//used to compare old nums to new nums
      final Agent data = this;//used to pass the identity of the calling data
      agent to the classifier for creation in the same container

      protected void setup() {
            System.out.println("Data Extractor Active "+ getLocalName());

            DFAgentDescription dfd = new DFAgentDescription();
            dfd.setName(this.getAID());
            addBehaviour(new DFRegistration(this, "Raw Data
                  Extraction","Frequency 5", dfd));//Describe Data service
                  and register with DF

            //Create a message template for the Contract Net Protocol
            MessageTemplate mt =
                  ContractNetResponder.createMessageTemplate(FIPANames.Intera
                  ctionProtocol.FIPA_CONTRACT_NET);
            /*Open socket and pass data to Classifier*/
            try {//opens port for receiving data stream
                  socket = new ServerSocket(9999);
                  //System.out.println(getLocalName()+":ServerSocket
                        created.");
            }

            catch(IOException e){System.out.println("ERROR1");}

            //System.out.println(getLocalName()+" About to add behaviours.");
            Behaviour loop = new TickerBehaviour (this, 1){
            protected void onTick(){//Action that periodically happens on
                  each tick
```

```
            addBehaviour(new DataSocketServer(socket){});
            //System.out.println("Data: creating Classifier Agent...");
            addBehaviour(new CreateClassifier(data){});
            //System.out.println("Adding Data Client behaviour to "+
                  getLocalName());
            addBehaviour(new DataClient(DataSocketServer.temp){});//passes
                  received data to client for Classifier to access
            //System.out.println("Finished adding Data Client behaviour to "+
                  getLocalName());
            }//end of onTick() method
            };

            addBehaviour(loop);

      }

}
```

Appendix G

DataClient Behaviour

```
/*DataClient behavior used Data Agent to access the data stream*/

package main;
import jade.core.behaviours.OneShotBehaviour;
import java.io.*;
import java.net.*;

public class DataClient extends OneShotBehaviour {//behaviour used to pass
      data to pass data to classifier
      Socket socket;
      PrintStream tpPS;
      String input;
      static boolean checker = false;//checks if behaviour were processed to
            completion

      public DataClient(){

      }
      public DataClient(String data ){
            input = data;
      }

      public void action(){
            checker = false;
            try{//attempt to pass data on the current machine port number
                  9997
                  //System.out.println("Accessing socket 9997.");
                  socket = new Socket("http://152.8.113.111", 8090);
                  //System.out.println("Successfully accessed socket 9997.");
                  tpPS = new PrintStream(socket.getOutputStream());
            } catch(Exception e){}

            //tpPS.println(DataSocketServer.temp);
            tpPS.println("Hello");
            tpPS.println("5");
            //System.out.println("Done receiving from client. Now passing to
                  Classifier Socket...");
            checker = true;

      }

}
```

Appendix H

DataSocketServer Behaviour

```
/*DataSocketServer used by Data Agent to open a socket and stream*/

package main;
import java.net.*;
import java.io.*;
import jade.core.behaviours.OneShotBehaviour;

public class DataSocketServer extends OneShotBehaviour{
	ServerSocket socket;
	static String temp;
	static BufferedReader ss_BR = null;
	Socket ss_accept = null;

	public DataSocketServer (ServerSocket ss, BufferedReader br){
		socket = ss;
		ss_BR = br;

		try {
			ss_accept = socket.accept();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	public DataSocketServer (ServerSocket ss){
		socket = ss;
		try {
			ss_accept = socket.accept();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

		try {
			ss_BR = new BufferedReader(new
				InputStreamReader(ss_accept.getInputStream()) );
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	public DataSocketServer (Socket soc){

		ss_accept = soc;
		try {
			ss_BR = new BufferedReader(new
				InputStreamReader(ss_accept.getInputStream()) );
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
```

```
		}
	}

	public void action() {
		try{
			do{
				temp = ss_BR.readLine();
	//			System.out.println(temp);
			}while(!ss_BR.readLine().isEmpty());
		}
		catch(SocketException e){}//
		catch (IOException e) {
			e.printStackTrace();
		}

	}

}
```

Appendix I

Feature Extraction Agent

```
/*Feature Extraction Agent code used to register with DF, participate in
      Monitor Agent initiated CNP and for pipeline with Classifier Agent*/

package main;

import java.io.IOException;
import main.Classifier.Looper;
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.*;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPANames;
import jade.domain.FIPAAgentManagement.*;
import jade.lang.acl.*;
import jade.proto.ContractNetResponder;

public class Feature extends Agent{

      MessageTemplate mt;
      protected String features;
      protected String service;
      AID monitor;
      AID classifier;

      //**
      public SequentialBehaviour seq;

      protected void setup(){

            service = "Feature Extraction";
            features = "a b";
            if(getArguments()!=null){Object[] args = getArguments(); features
      = args[0].toString();}

            DFAgentDescription dfd = new DFAgentDescription();
            dfd.setName(this.getAID());

            addBehaviour(new DFRegistration(this, service , features, dfd));

            seq = new SequentialBehaviour();/*MessageTemplate, CNPResponse,
                  ClassifierBehaviour */

           seq.addSubBehaviour(new OneShotBehaviour() {
                  public void action (){
                        mt =
                              ContractNetResponder.createMessageTemplate(FIPA
                              Names.InteractionProtocol.FIPA_CONTRACT_NET);}}
                              );

            seq.addSubBehaviour(new CNPResponse(this, mt));
```

```
	seq.addSubBehaviour(new OneShotBehaviour(){
		public void action (){
			ACLMessage receiveMsg = receive();

			try {//pulls awarded feature extractor's AID and uses
				that to send messages
				AID[] aid= new AID[2];

				while (true){
					if (receiveMsg !=null){
						aid = (AID[])
						receiveMsg.getContentObject();
							ACLMessage informMsg = new
							ACLMessage(ACLMessage.INFORM;
						informMsg.addReceiver(aid[1]);
						informMsg.setContentObject(aid[0]);
						break;
					}
					else continue;
				}

			} catch (UnreadableException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			} catch (IOException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
		}
	});


	Looper loop = new Looper(this,seq);

}
class Looper extends CyclicBehaviour{//CyclicBehaviour {
	Behaviour bev;
	public Looper(Agent a, Behaviour b) {
		super(a);
		addBehaviour(b);
		bev = b;
	}

	public void action() {}//myAgent.removeBehaviour(bev); }

	}

	public String getService(){return features;}

	public void wonCNP(AID m, AID c){
		monitor = m;
		classifier = c;

		addBehaviour(new OneShotBehaviour(){
	public void action(){
```

```
                        ACLMessage confirm = new
                              ACLMessage(ACLMessage.CONFIRM);
                        confirm.addReceiver(monitor);
                        confirm.setContent("Got it working smooth");
                        myAgent.send(confirm);

                  }
            });

            addBehaviour(new OneShotBehaviour(){
                  public void action(){
                        ACLMessage confirm = new
                              ACLMessage(ACLMessage.AGREE);
                        confirm.addReceiver(classifier);
                        try {
                              confirm.setContentObject(monitor);
                        } catch (IOException e) {
                              // TODO Auto-generated catch block
                              e.printStackTrace();
                        }
                        myAgent.send(confirm);

                        Looper loop = new Looper(myAgent,seq);

                  }
            });
      }

      public void setMonitor(AID m){ monitor = m;}

}
```

Appendix J

Classifier Agent

```
/*Classifier Agent code used to register with DF, participate in Monitor
      Agent initiated CNP and for pipeline with Feature Extraction Agent*/

package main;

import java.io.*;
import java.net.*;
import javax.swing.JOptionPane;
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.*;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPANames;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.lang.acl.*;
import jade.proto.ContractNetResponder;

public class Classifier extends Agent {
      ServerSocket socket, kMeanS=null;
      Socket ss_accept=null;
      String input;
      MessageTemplate mt;
      protected String service;
      protected String featuresReq;
      public SequentialBehaviour seq;
      protected void setup(){

            service = "kMeans 5";
            featuresReq = "a b";
            if(getArguments()!=null){
                  Object[] args = getArguments();
                  service = args[0].toString(); featuresReq =
                        args[1].toString();
            }
            DFAgentDescription dfd = new DFAgentDescription();
            dfd.setName(this.getAID());

            addBehaviour(new DFRegistration(this, "Classification", service,
                  dfd));

            seq = new SequentialBehaviour();/*MessageTemplate, CNPResponse,
                  ClassifierBehaviour */

            seq.addSubBehaviour(new OneShotBehaviour() {
                  public void action (){
                        mt=ContractNetResponder.createMessageTemplate(FIPANam
                              es.InteractionProtocol.FIPA_CONTRACT_NET);}});

            seq.addSubBehaviour(new CNPResponse(this, mt));
```

```
      Looper loop = new Looper(this,seq);
}//end of setup

class Looper extends CyclicBehaviour{//
      Behaviour bev;
    public Looper(Agent a, Behaviour b) {
      super(a);
      addBehaviour(b);
      bev = b;
    }

    public void action() {}//myAgent.removeBehaviour(bev); }

  }

public String getService(){return service;}
public String getFeatures(){return featuresReq;}

public void wonCNP(AID a){//method called by Monitor agent when
      classifier wins the bid in CNP
      final AID aid = a;
      //behaviour run to pick up response from Monitor
      addBehaviour(new OneShotBehaviour(){
            public void action(){
                  System.out.println("\n\n");
                  ACLMessage confirm = new
                        ACLMessage(ACLMessage.CONFIRM);
                  confirm.addReceiver(aid);
                  confirm.setContent(getFeatures());
                  myAgent.send(confirm);
                  myAgent.addBehaviour(new OneShotBehaviour(){
                        public void action (){

                              ACLMessage receiveMsg =
                                    blockingReceive(MessageTemplate.Mat
                                    chPerformative(ACLMessage.AGREE));

                              AID monitor;

                              if (receiveMsg != null)
                              {
                                    try {//pulls awarded classifier's
                                          AID and uses that to send
                                          message from Feature
                                          Extractor
                                          System.out.println(myAgent.ge
                                                tLocalName() + ":
                                                ACLMessage response...
                                                ");

                                          monitor = (AID)
                                                receiveMsg.getContentOb
                                                ject();//AID of the CNP
                                                initiating Monitor
```

```
                                    System.out.println(myAgent.ge
                                          tLocalName() + "
                                          received message: " +
                                          monitor);

                                    ACLMessage informMsg = new
                                          ACLMessage(ACLMessage.A
                                          GREE);

                                    informMsg.addReceiver(monitor
                                          );

                                    informMsg.setContent("CNP
                                          Process Completed. "+
                                          getLocalName() +"
                                          working with "+
                                          receiveMsg.getSender().
                                          getLocalName());
                                    myAgent.send(informMsg);
                                    //System.out.println("Sent
                                          "+informMsg.getContent(
                                          ));
                                    //System.out.println("CNP
                                          Cycle Complete");

                                                      block();
                              } catch (UnreadableException e) {
                                    // TODO Auto-generated catch
                                          block
                                    e.printStackTrace();
                              }
                        }
                        else
                        {
                              block();
                        }
                  }});
            }
      });
}
}
```

Appendix K

CNPResponse Behaviour

```
/*CNPResponse behavior used by Classifier Agent and Feature Extractor Agent
      to respond to the Monitor Agent's CallForProposal*/

package main;

import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.lang.acl.UnreadableException;
import jade.proto.ContractNetResponder;
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.OneShotBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.RefuseException;
import jade.domain.FIPAAgentManagement.NotUnderstoodException;

public class CNPResponse extends ContractNetResponder{
      protected AID monitor;
      public CNPResponse(Agent a, MessageTemplate mt) {super(a, mt);
      System.out.println(myAgent.getLocalName() +": Starting
      CNPResponse...");}

      protected ACLMessage prepareResponse(ACLMessage msg) throws
      NotUnderstoodException, RefuseException
      {
            monitor = msg.getSender();
            System.out.println(myAgent.getLocalName()+": __"+"prepared a
      response...**\n");

            if (myAgent.getClass().toString().equals("class
      main.Classifier")){
                  System.out.println("Made It In!!!");
                  Classifier agent = (Classifier)myAgent;
                  //Sends bid to Monitor agent

                  ACLMessage response = msg.createReply();
                  response.setPerformative(ACLMessage.PROPOSE);
                  response.setContent(agent.getService());//replies to
      Monitor agent with the services it offers as a bid
                  System.out.println(myAgent.getLocalName() + " CNPResponse
      finished response");
                  return response;
            }
            else {
                  System.out.println("Made It In***");
                  Feature agent = (Feature) myAgent;
                  //Sends bid to Monitor agent
                  System.out.println("cfp received: " + msg.getContent());
                  ACLMessage response = msg.createReply();
                  response.setPerformative(ACLMessage.PROPOSE);
```

```
                response.setContent("" + agent.getService());
                System.out.println(myAgent.getLocalName()+" CNPResponse
        finished");
                return response;
          }

        }//end of ACLMessage prepareResponse(ACLMessage msg) block

        protected ACLMessage prepareResultNotification( ACLMessage cfp,
        ACLMessage propose, ACLMessage accept)
        {
          //Accepts awarded contract
          if (myAgent.getClass().toString().equals("class
                main.Classifier")){

                System.out.println(myAgent.getLocalName()+" acceptance
                      received: " + accept.getContent());
                ACLMessage response = new ACLMessage
                      (ACLMessage.INFORM);//accept.createReply();
                response.setPerformative(ACLMessage.INFORM);
                response.setContent("Ready to go");
                Classifier agent = (Classifier) myAgent;
                agent.wonCNP(monitor);
                return response;
          }

          else{
                try {
                      System.out.println(myAgent.getLocalName()+"
                      acceptance received: " + accept.getContentObject());
                } catch (UnreadableException e1) {
                      e1.printStackTrace();
                }
                ACLMessage response = new ACLMessage
                      (ACLMessage.INFORM);//accept.createReply();
                response.setPerformative(ACLMessage.INFORM);
                response.setContent("Ready to go");
                Feature agent = (Feature) myAgent;
                agent.setMonitor(monitor);
                try {
                      agent.wonCNP(monitor, (
                            AID)accept.getContentObject());
                } catch (UnreadableException e) {

                      e.printStackTrace();
                }
                return response;
          }
        }//end of protected ACLMessage prepareResultNotification block

        protected void handleRejectProposal(ACLMessage cfp, ACLMessage propose,
          ACLMessage reject) {
          //Contract Rejected
          System.out.println("rejection received: " + reject.getContent());
        }//end of protected void handleRejectProposal block
}
```

Appendix L

CreateClassifier Behaviour

```
/*CreateClassifier behavior used to allow an agent to create a new Classifier
      Agent*/

package main;
import jade.core.Agent;
import jade.core.behaviours.OneShotBehaviour;
import jade.wrapper.AgentController;
import jade.wrapper.PlatformController;
import javax.swing.tree.DefaultMutableTreeNode;

public class CreateClassifier extends OneShotBehaviour {
      static int countClassifier = 0;
      Agent agent;
      public CreateClassifier(Agent a){
            agent = a;
      }
      @Override
      public void action() {
            // TODO Auto-generated method stub
            try{//create a new monitor agent in the main container
                  PlatformController container =
                  agent.getContainerController();
                  AgentController agent =
                        container.createNewAgent("Classifier_"+countClassifie
                              r, "main.Classifier", null);
                  agent.start();
                  countClassifier++;
            }catch(jade.wrapper.ControllerException err){
                  System.out.println("ERROR CREATING NEW AGENT: " + err);
            }//ends try/catch(jade.wrapper.ControllerException err){
      }
}
```

Appendix M

ClassifierClient Behaviour

```
/*ClassifierClient behaviour used to test flow of data communicated between
via sockets*/


package main;
import jade.core.behaviours.OneShotBehaviour;

import java.io.*;
import java.net.*;

public class ClassifierClient extends OneShotBehaviour {//behaviour used to
pass data to pass data to classifier
      Socket socket;
      PrintStream tpPS;
      String input;

      public ClassifierClient(){
            action();
      }
      public ClassifierClient(String data ){
            input = data;
            action();
      }

      public void action(){

            try{//attempt to pass data on the current machine port number
                  9995
                  socket = new Socket("localhost", 9995);
                  tpPS = new PrintStream(socket.getOutputStream());

            } catch(Exception e){System.out.print(e);}
                  tpPS.println(input);
      }

}
```

Appendix N

ClassifierSocketServer Behaviour

/*ClassifierClient behaviour used to test flow of data communicated between via sockets*/

```
package main;
import java.net.*;
import java.io.*;

import jade.core.behaviours.OneShotBehaviour;

public class ClassifierSocketServer extends OneShotBehaviour{
	ServerSocket socket;
	static String temp;
	static BufferedReader ss_BR = null;
	Socket ss_accept = null;
	static boolean checker = false;

	public ClassifierSocketServer (ServerSocket ss, BufferedReader br){
		socket = ss;
		ss_BR = br;

		try {
			ss_accept = socket.accept();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	public ClassifierSocketServer (ServerSocket ss){
		socket = ss;
		try {
			ss_accept = socket.accept();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			System.out.println(e);
		}

		try {
			ss_BR = new BufferedReader(new
				InputStreamReader(ss_accept.getInputStream()) );
		} catch (IOException e) {
			System.out.println(e);
		}
		action();
	}

	public ClassifierSocketServer (Socket soc){

		ss_accept = soc;
		try {
			ss_BR = new BufferedReader(new
				InputStreamReader(ss_accept.getInputStream()) );
		} catch (IOException e) {
```

```
                // TODO Auto-generated catch block
                System.out.println(e);
            }
        }

        public void action() {
            checker = false;
            try{

                temp = ss_BR.readLine();
                checker = true;
            }
            catch(SocketException e){System.out.println(e);}//
            catch (IOException e) {
                System.out.println(e);
            }

        }//end of action method


}
```

Appendix O

DFRegistration Behaviour

```
/*DFRegistration behavior is used by all agents that need to register their
      services with the DF*/

package main;
import jade.core.Agent;
import jade.core.behaviours.OneShotBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPANames;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.SearchConstraints;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.lang.acl.MessageTemplate;
import jade.proto.ContractNetResponder;

public class DFRegistration extends OneShotBehaviour{
      Agent agent;
      String service1;
      String service2;
      DFAgentDescription dfd;

      public DFRegistration(Agent a, String s, String ss, DFAgentDescription
            d){
            agent = a;
            service1 = s;
            service2 = ss;
            dfd = d;
      }
      public void action() {

            ServiceDescription servDesc = new ServiceDescription();
            servDesc.setName(service1);
            servDesc.setType(service1);
            ServiceDescription servDesc2 = new ServiceDescription();
            servDesc2.setName(service2);
            servDesc2.setType(service2);
            try{
                  dfd.addServices(servDesc);
                  dfd.addServices(servDesc2);
                  //register Data Agent service with DF
                  DFService.register(agent, dfd);
            }catch(FIPAException e){
                  System.err.println(e.getMessage());
            }
            catch (Exception e){
                  System.out.println(e);
            }
      }
}
```